Learning Weakly Correlated Cause-Effects for Gardening with a Cognitive System

Alejandro Agostini[a,*], Carme Torras[b], Florentin Wörgötter[a]

[a]*Bernstein Center for Computational Neuroscience, 37073 Göttingen, Germany*
[b]*Institut de Robòtica i Informàtica Industrial (CSIC-UPC), 08028 Barcelona, Spain*

Abstract

We propose a cognitive system that combines artificial intelligence techniques for planning and learning to execute tasks involving delayed and variable correlations between the actions executed and their expected effects. The system is applied to the the task of controlling the growth of plants, where the evolution of the plant attributes strongly depends on different events taking place in the temporally distant past history of the plant. The main problem to tackle is how to efficiently detect these past events. This is very challenging since the inclusion of time could make the dimensionality of the search space extremely large and the collected training instances may only provide very limited information about the relevant combinations of events. To address this problem we propose a learning method that progressively identifies those events that are more likely to produce a sequence of changes under a plant treatment. Since the number of experiences is very limited compared to the size of the event space, we use a probabilistic estimate that takes into account the lack of experience to prevent biased estimations. Planning operators are generated from most accurately predicted sequences of changes. Planning and learning are integrated in a decision-making framework that operates without task interruptions by allowing a human gardener to instruct the treatments when the knowledge acquired so far is not enough to make a decision.

Keywords: decision-making framework, planning operator learning, probability estimate, weakly correlated cause-effects, growth of plants

*Corresponding author.
Email addresses: aagostini@physik3.gwdg.de (Alejandro Agostini), torras@iri.upc.edu (Carme Torras), woergoetter@physik3.gwdg.de (Florentin Wörgötter)

1. Introduction

This work addresses the complex problem of how to control the evolution of a system where the relations between actions applied to the system and their corresponding effects take place with variable delays and strengths. In particular, we will focus on the application of controlling the growth of plants, which is an industrially relevant application useful to control, for example, seed productions and plant breeding. Controlling the growth of plants is a challenging problem requiring advanced predictive cognitive properties, which so far can only be provided by experienced human gardeners (GARNICS, 2010-2013).

In this work we propose a cognitive system to control the growth of plants using artificial intelligence (AI) techniques for planning and learning. The main reason behind this is that controlling the growth of plants is indeed a human-like task that can be easily formulated in a declarative language, compatible with AI techniques. This allows using robust and well-known AI techniques for reasoning and decision-making such as logic-based planning (LaValle, 2006). This kind of planners uses a set of planning operators coding cause-effects in terms of propositions and logical predicates easily understandable by humans, making the definition of human-related decision-making problems, such as controlling the growth of plants, straightforward. However, the hand-coding of the planning operators may be very complicated. Even for apparently simple tasks, many of the relevant aspects of the cause-effects to be encoded can be easily overlooked. This is particularly true in applications involving weakly correlated cause-effects since it is difficult to precisely specify the expected evolution of plants and the past events responsible for this evolution. Therefore, to avoid spending a lot of time in coding such operators and to prevent long task interruptions due to an incomplete or incorrect coding of the operators, the system should learn them online, while the task is executed, and from the fewest experiences possible.

The requirement of online learning of operators is achievable if planning is interleaved with learning so as to progressively improve the decision-making capabilities of the planner by injecting new knowledge after each action execution. Interleaving planning and learning has been proposed in several works to improve the performance of a logic-based planner (Agostini et al., 2011; Wang, 1995, 1996; Benson, 1995; Shen, 1989; Gil, 1994; Walsh and Littman, 2008). For instance, in the works by Wang (1995, 1996) a set of planning operators is initially generated using examples collected offline from expert's solution traces of the task to be executed. Then, the system refines these operators using experiences collected from a simulated environment where a modified PRODIGY planner is used

for decision-making, and an adaptation of the Version Space method (Mitchell, 1997) called the OBSERVER, is used to refine planning operators. If the planner fails in finding a plan due to an incomplete operators set, it returns a failure signal and interrupts the execution of the task. In Benson (1995) a strategy called TRIAL for continuous planning and learning is proposed. TRIAL continuously generates plans, executes them, and learns operators until the system reaches a planning impasse, in which case an external teacher is called to take control of the task and complete it. In Gil (1994) a system called EXPO uses PRODIGY as a baseline planner and improves its domain knowledge assuming it is completed up to 50 %. Planning operators are learned using results obtained from simulated environments and they become immediately available for planning. We would like to remark that none of the aforementioned approaches has been developed for tasks involving weakly correlated cause-effects.

We present a decision-making framework that interleaves AI techniques for planning and learning of weakly correlated cause-effects to provide a cognitive system with decision-making capabilities for controlling the growth of plants (GARNICS, 2010-2013). In this task, the system should make decisions about which treatment, e.g. which combination of water and nutrient, a plant should receive to achieve a given goal state provided the history of the plant. In order to implement our cognitive system we use the framework proposed in Agostini et al. (2011) as the basic architecture to integrate planning and learning. The framework operates without task interruptions by allowing a human expert to make a decision in those situations where the knowledge acquired so far is not enough to decide the action autonomously. We adapt this framework for tasks involving weak correlations and propose a new method for learning planning operators coding weakly correlated cause-effects.

The main idea of learning weakly correlated cause-effects is to identify those events (sensed values and actions) that once observed in the past history of a plant permit predicting the future evolution of the plant attributes. There are two important difficulties to face in this regard. The first one is the dimensionality of the sequence space, where the possible combinations of events is infinite, provided we assume sequences of any length. For example, given an M-dimensional attribute space and an action space of dimensionality N, at each time step there are $N+M$ dimensions to explore. If we want to evaluate events belonging to 10 different time steps, then the dimensionality of the search space is $10\text{x}(N+M)$. The second difficulty is the limited number of training instances available (with respect to all the possible combinations of attribute, actions, and time-steps). This is a problem since cause-effects rules should correctly predict the future evolution

of attributes during sequences never experienced before. To cope with these problems the learning method should be able to generate cause-effects that perform extensive generalization, in feasible time, and using few experiences.

To address the learning problem we use a parallel strategy that generates many alternative combinations of past events that are potentially relevant to predict the sequence of changes obtained from the execution of a sequence of actions. From all these alternative combinations, the one that most likely produces the sequence of changes is used for planning. Our approach could be compared to expanding multiple independent search trees that are not subordinated to one another. This permits trying in parallel combinations of events from many different regions of the event space (possibly disjoint), increasing the chances of finding combinations with accurate predictions.

Not all the possible sequence of changes may be important to arrive at a good control of the growth of the plant. Therefore, to speed up the learning, we let the system learn planning operators mostly coding important sequences of changes by using the help of a human gardener that guides the exploration of actions towards those producing relevant changes. To have confident evaluations of combinations of events from a small number of observations, we use the probabilistic estimate proposed in Agostini et al. (2011), the *density*-estimate. This estimate takes into account the lack of experience to compensate the bias introduced when the estimation is done from a few examples.

The layout of the paper is as follows. The next section introduces the general notation. Section 3 outlines the elements of the decision-making framework. Later, in Section 4, we provide the details of the mechanisms for learning planning operators coding weakly correlated cause-effects. Section 5 presents the task of controlling plant growth. In Section 6, we present the results of the application of our cognitive system to such task. In this section we also provide a thorough comparison with one of the most widely used methods for learning planning operators, the OBSERVER method (Wang, 1995). The paper ends with conclusions about the work.

2. Basic Notation

We assume that there is a finite (or countable infinite) set of world states $\mathscr{S}$ and a finite (or countable infinite) set of actions $\mathscr{A}$ executable at those states. A state $s \in \mathscr{S}$ is described using a set of descriptors $\{d_i\}$, $i = 1...N$, which can take different discrete values $d_{i,j}$, $j = 1...|d_i|$. We use the notation a_k, $k = 1...|\mathscr{A}|$, to refer to a particular action.

To provide a concrete example of possible instantiations of descriptors, we use the state-space definition for the task of controlling the growth of plants presented in Table 2 (Sec. 5). For instance, the first descriptor, d_1, corresponding to the number of leaves of the plant, can take different discrete values:

$$d_{1,j} \in \{0,1,2,3,4,5,6,7,8,9,10,>10\}, \tag{1}$$

with $j = 1,...,12$.

2.1. Events

We use the notation s^t to indicate a state observed at time-step t and the notation a^t to refer to an action taken at time-step t (wrt a given time reference t=0). In the same way, we will use $d^t_{i,j}$ and a^t_k to indicate the instantiations of descriptor d_i and the action executed at time t, respectively. We will refer to an event at time t, v^t, as either a value of the descriptor $d^t_{i,j}$ or an action a^t_k.

2.2. Sequence Space

We refer to a *sequence* as any sequence of states and actions of any length. The (infinite) set of all the possible sequences configures the *sequence space*. We define a *subspace q* of sequences, or simply a subspace, as a set of events

$$q = \{v^{t_i}, v^{t_{i+k}}, ..., v^{t_{i+m}}\}, \tag{2}$$

where the superscripts may refer to the same or different time-steps. The subspace represents all the sequences having the enumerated events, hence permitting a compact representation suitable for generalization over sequences. For instance, the subspace

$$q_1 = \{d^{-1}_{1,2}, d^0_{1,3}, d^0_{2,3}, d^1_{1,4}\} \tag{3}$$

covers all the sequences having the second value of descriptor 1, one time-step before a reference time $t = 0$, the third value of descriptor 1 and the third value of descriptor 2, both at time $t = 0$, and the fourth value of descriptor 1, one time-step after the reference time.

To shed more light on the example sequence, we use again the state-space definition of Table 2. Using this definition, the subspace q_1 would code an observation of 1 leaf one time-step before $t = 0$ ($d^{-1}_{1,2}$), 2 leaves at $t = 0$ ($d^0_{1,3}$), all of them green ($d^0_{2,3}$), and 3 leaves one time-step after $t = 0$ ($d^1_{1,4}$).

2.2.1. Subspace Distance

We define the distance between two subspaces q and u as

$$\text{diff}(q,u) = \frac{\sum_{t=t_{min}}^{t_{max}} D^t_{q,u}}{t_{max} - t_{min}}, \tag{4}$$

where t_{min} and t_{max} is the minimum and maximum time-steps in q and u, and $D^t_{q,u}$ is the typical metric used in nearest neighbour algorithms (Finnie and Sun, 2002),

$$D^t_{q,u} = 1 - \frac{\sum_{i=1}^{N} w_i \, sim(v^t_{i,q}, v^t_{i,u})}{\sum_{i=1}^{N} w_i}, \tag{5}$$

where N is the number of state descriptors, w_i is the importance weighting of descriptor i, and $sim(v^t_{i,q}, v^t_{i,u})$ is the similarity measure between events $v^t_{i,q}$ and $v^t_{i,u}$, of the subspaces q and u, respectively. If the variable corresponding to an event has ordinal values, the similarity is calculated as

$$sim(v^t_{i,q}, v^t_{i,u}) = 1 - \frac{|j-k|}{M} \tag{6}$$

where M is the number of possible values of the variable, and j and k are the indexes of the ordinal values $v^t_{i,q}$ and $v^t_{i,u}$, respectively. On the other hand, if the descriptor is defined in a non-ordinal way, the similarity between the two events is defined as $sim(v^t_{i,q}, v^t_{i,u}) = 1$, if $j = k$, and $sim(v^t_{i,q}, v^t_{i,u}) = 0$, otherwise.

Note that events $v^t_{i,q}$ and $v^t_{i,u}$ can be state descriptors or actions. A descriptor at a specific time-step in one subspace would be only comparable with the same descriptor, though with likely a different value, at the same time-step of the other subspace. The same is applicable for the comparison between actions. On the other hand, if a descriptor or action of one subspace is not present in the other one, the similarity measure is 1. This is so since, if an event is not considered in one of the subspaces, it indicates that all the values for that event are actually covered by that subspace, including the one of the other subspace. For example, using the state-space definition of Table 2 to define the value of M, the similarity measures between events of subspaces

$$q = \{d^{-2}_{1,2}, d^{-1}_{1,3}, d^{0}_{2,3}\}, \tag{7}$$

and

$$u = \{d^{-2}_{1,4}, d^{0}_{2,3}\}, \tag{8}$$

would be

$$sim(d_{1,2}^{-2}, d_{1,4}^{-2}) = 5/6, \tag{9}$$

$$sim(d_{1,3}^{-1}, \emptyset) = 1, \tag{10}$$

and

$$sim(d_{2,3}^{0}, d_{2,3}^{0}) = 1, \tag{11}$$

where $\emptyset$ denotes that the corresponding descriptor is not present in subspace u at time-step -1.

2.3. Rule Representation

Rules coding weakly correlated cause-effects are represented using the conventional notation of precondition-action-effects. However, there are some fundamental differences between the traditional representation of cause-effect rules and the representation of cause-effects coding weak correlations. In the latter case, each part of a rule consists not of events at a single time-step, but of sequences of events where time is now explicitly represented.

Each rule has a reference time $t = 0$ that defines past and future events. The precondition part considers events in the past, both descriptors and actions, and events consisting only of descriptors at the reference time $t = 0$. The action part consists of actions applied from the reference time $t = 0$ on. Finally, the effect part is composed of events consisting only of descriptors in future time steps.

To refer to a rule we use the notation

$$r = \{P, A, E\}, \tag{12}$$

where P, A, and E are the precondition, action, and effect parts of the rule, respectively. To illustrate the rule codification, one instantiation of the parts of a rule could be

$$\begin{aligned} P &= \{d_{1,1}^{-1}, a_1^{-1}, d_{1,3}^{0}\}, \\ A &= \{a_1^0, a_1^1\}, \\ E &= \{d_{1,5}^{1}, d_{1,7}^{2}\}. \end{aligned} \tag{13}$$

Note that the actions executed in the past become conditions in the precondition part. Using again the state-space definition of Table 2 to give a concrete example of descriptor instantiations, rule (13) would code the evolution of the number of leaves (d_1), starting from 2 leaves at $t = 0$ ($d^0_{1,3}$), following with 4 leaves in the next time-step ($d^1_{1,5}$), and ending with 6 leaves in the following time-step ($d^2_{1,7}$). The rule also considers in its precondition part two past events that would be necessary to obtain the coded changes: 0 leaves should be observed in the plant one time-step earlier than $t = 0$ ($d^{-1}_{1,1}$) and the treatment a_1 of mild watering (see Table 4) should have also been applied one time-step earlier.

A rule represents all the sequences in which the coded weak correlation takes place. To see this more clearly, let us define a subspace

$$q = \{P \cup A \cup E\}. \tag{14}$$

Then, all the sequences covered by the subspace q contain the events that code the same weakly correlated cause-effects. Ideally, we would like to learn rules that permit good predictions of the effects, i.e. that the sequences included in the subspace $\{P \cup A\}$, once experienced, lead to the same effect E.

2.4. Instances

The experienced sequences are stored as instances $\hat{s}^t$, consisting in the tuple

$$\hat{s}^t = (\hat{s}^{t-1}, a^{t-1}, s^t), \tag{15}$$

which describes all the state transition information up to time t, where s^t is the state description at time t, a^{t-1} is the action executed at $t-1$, and $\hat{s}^{t-1}$ consists of the past experienced sequence of state-action-state transitions until time-step $t-1$.

We say that an instance is covered by a rule $r = \{P, A, E\}$ when $q_{PA} = \{P \cup A\}$ matches the instance. If a covered instance also matches $q_{PAE} = \{P \cup A \cup E\}$ we denote it as a *positive* instance, otherwise we call it a *negative* instance.

2.5. Coding Weakly Correlated Cause-Effect into Planning Operators

Logic-based planners assume Markov decision process models of the environment, i.e. the probability of transition between states only depends on the state at which the action is executed. The representation of state transitions is carried

out through the planning operators (POs), which compactly represent the transitions by only considering the changes in the state with an action as well as all the non-changed descriptors that are responsible for these changes [1].

However, tasks involving weakly correlated cause-effects are essentially non-Markovian since the probability of transitions between states depends not only on the state in which the action is executed but also on previous states. To permit coding weakly correlated cause-effects into logic-based POs we use the well known fact that a non-Markovian decision process can be transformed into a Markovian one by a redefinition of the state considering history (McCallum, 1996): the current state would consist not only of the current sensory input, but also of past sensory inputs and actions. To do this, we define a state for planning using the instance definition of previous Sec. 2.4, rewritten here for convenience,

$$\hat{s}^t = (\hat{s}^{t-1}, a^{t-1}, s^t). \tag{16}$$

This state definition fulfils the Markovian property since

$$\Pr(\hat{s}^{t+1}|\hat{s}^t, a^t, \hat{s}^{t-1}, a^{t-1}, ...) = \Pr(\hat{s}^{t+1}|\hat{s}^t, a^t). \tag{17}$$

To represent a logic-based PO we use the traditional STRIPS-like representation (LaValle, 2006) formed by the precondition part, and the addition (events added to the state with the action) and deletion (events deleted from the state with the action) parts to code the effects.

To generate a plan, the planner evaluates sequences of POs so as to transform the current state into any of the goal states, by adding and deleting descriptors from the state representation according to the addition and deletion parts of the evaluated POs, respectively. In our case, the current state codes all the past events up to time $t = 0$. Therefore, in order to add and delete events from the current state, as well as to evaluate which PO to apply at each situation, the events of the precondition, addition, and deletion parts should also be referenced up to time $t = 0$. This is implemented by including in the addition part all the events coded in the rule, shifting the time reference such that the last event in the rule coincides with time $t = 0$. The deletion part suffers no time shift and includes all the events that should be deleted from the state, i.e. the events of the precondition part of

[1] To rapidly provide an example of a non-changing but causative descriptor, imagine a PO coding the action of opening a door. In this case, a non-changed causative descriptor could be a predicate indicating that the door is unlocked, which would actually permit the door opening and remains unchanged after executing the action.

the rule that are no longer in the new state due to the time shift. Finally, the precondition part of the rule is considered as such for the precondition part of the PO since it already matches the time reference of the current state.

To provide a clarifying example on how the coding of a PO is carried out from a cause-effect rule, we use the example rule (13). Note that this rule codes a two-step effects: the plant reaching 4 leaves one time-step after the initiation of the rule execution ($d^1_{1,5}$) and 6 leaves two time-steps after the application of the rule ($d^2_{1,7}$). To code them into the addition part of the rule, these two events should be referenced with respect to $t = 0$, indicating that the plant would have 6 leaves after the PO execution ($d^0_{1,7}$) and would have had 4 leaves one time-step before ($d^{-1}_{1,5}$). The same is applicable to all the other events coded in the rule. For example, after the rule execution, the events in the precondition part of the rule would be $\{d^{-3}_{1,1}, a^{-3}_1, d^{-2}_{1,3}\}$ while those from the action part would be $\{a^{-2}_1, a^{-1}_1\}$. All these events are considered in the addition part of the PO since they should be added to the new state. After completing the precondition and deletion parts according to the previous explanations, the PO generated from the cause-effect rule (13) would be

$$\begin{aligned}
&\text{name} = r,\\
&\text{prec} = d^{-1}_{1,1} \wedge a^{-1}_1 \wedge d^0_{1,3},\\
&\text{del} = \{d^{-1}_{1,1}, a^{-1}_1, d^0_{1,3}\},\\
&\text{add} = \{d^{-3}_{1,1}, a^{-3}_1, d^{-2}_{1,3}, d^{-1}_{1,5}, d^0_{1,7}, a^{-2}_1, a^{-1}_1\},
\end{aligned} \tag{18}$$

where the precondition part (prec) has the same events of the precondition part of the rule, but now coded in a conjunctive normal form, as required by the STRIPS-like coding, the addition (add) and deletion (del) parts contains the events with the corresponding time shift, and r is the name assigned to the PO.

2.6. Elements for Planning

The elements used by a logic-based planner to generate a plan consist of a set of objects, a set of predicates, describing properties of the objects or relations between them, a description of the initial state for planning $\hat{s}_{ini}$, from which the planner should find a plan to achieve a goal state s_g, a goal specification g, consisting of the set of predicates describing all the goal states $\{s_g | g \subseteq s_g\}$, and a set of planning operators (POs) used to build plans. In the problem of planning in weakly correlated environments the initial state for planning $\hat{s}_{ini}$ is formed by

all the events describing the current as well as all the previous states and actions taken on the plant.

3. Decision-Making Framework

To provide the agent with the capability of decision-making we implemented the decision-making framework depicted in Fig. 1. In this framework, the learner constantly enriches the PO set with every experienced instance. With the operators learned up to a given moment, the planner tries to find a plan that would permit achieving the goal (provided by the user) from the initial state. The state representation is generated with the help of the *perception* module that transforms the raw information of the sensors into declarative state descriptors. At each iteration, the planner yields one of the three possible outcomes: the first PO of a plan for its execution, an action request, or an end-of-plan (goal reached) signal. The action request takes place if the planner cannot generate a plan due to missing POs. In this case, the planner sends a request to a human expert (e.g. a gardener) which instructs the sequence of actions to be executed, allowing for the continuation of the task.

The actions provided either by the planner or the teacher are then sent to the *execution* module for their actual execution. After the execution, the planner generates a new plan and the process starts over again. This is a strategy known as *replanning*, which permits an immediate use of new POs generated by the learner and allows the system to find a new plan after eventual plan failures produced, for instance, by an incomplete PO set.

The algorithmic description of the mentioned process is shown in Alg. 1.

Algorithm 1 Decision-Making Framework

Specify the goal g
Initialize the rule set $\mathscr{R}$ (e.g. $\mathscr{R} \leftarrow \varnothing$)
$s^{t-1} \leftarrow \varnothing$
$a^{t-1} \leftarrow \varnothing$
$A \leftarrow \varnothing$ {empty action sequence}
repeat
  Get current state s^t {Perception module}
  Generate training instance $\hat{s}^t = (\hat{s}^{t-1}, a^{t-1}, s^t)$
  $\mathscr{R} \leftarrow$ LEARNER$(\mathscr{R}, \hat{s}^t)$ {update rule set with current instance}
  $\hat{s}_{ini} \leftarrow \hat{s}^t$
  if $g \subseteq \hat{s}_{ini}$ **then**
    End of plan
  else
    if action sequence A completed **then**
      $\mathscr{R}_{plan} =$ PLANNER $(\hat{s}_{ini}, \mathscr{R}, g)$ {find a plan to achieve goal g from current state $\hat{s}_{ini}$}
      if $\mathscr{R}_{plan} = \varnothing$ **then**
        $A \leftarrow$ TEACHER $(\hat{s}_{ini}, g)$ {request an action sequence to the teacher for the ongoing task}
      else
        $A = \mathscr{R}_{plan}(1).A$ {sequence of actions coded in the first PO of the plan}
      end if
    end if
    $a^t \leftarrow$ Next action in the sequence A
    Execute a^t {Execution module}
    $t \leftarrow t+1$
  end if
until End of plan

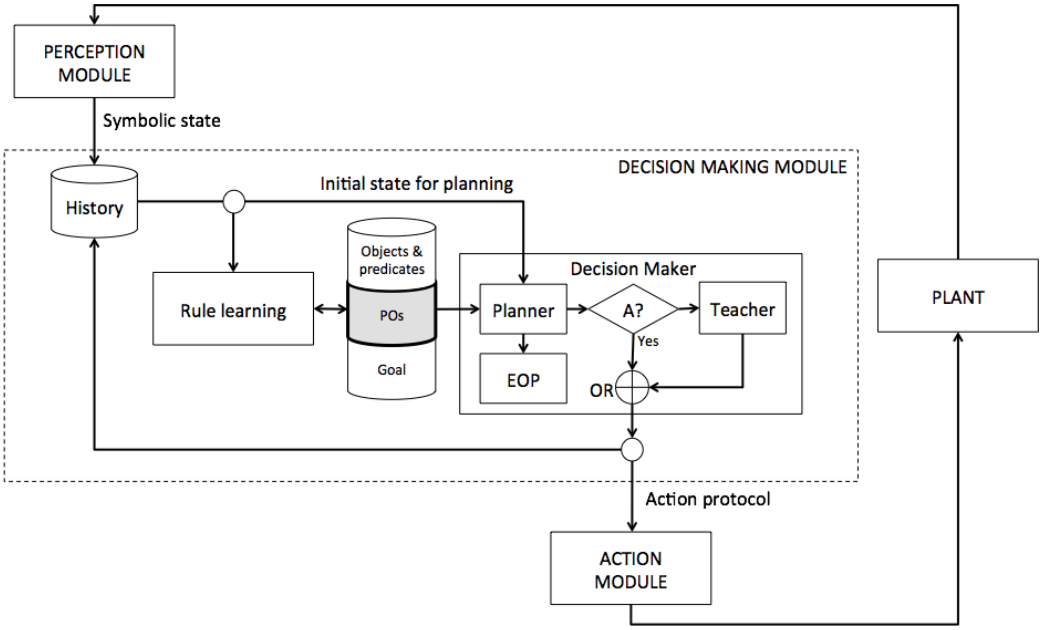


Figure 1: Schema of the decision-making framework.

4. The Learning Mechanisms

The role of the learning method is to find those events that should be observed in past states so as to be able to predict the changes that will be produced in future states when a sequence of actions is executed. In this process, it is important to keep the number of events as small as possible since the fewer the number of events, the larger the sequences represented in the sequence space and the better the generalization attained.

When relevant combinations of events are learned from scratch, the uncertainty on which combinations of events are relevant is very high and all the possible combinations should have a chance to occur. This poses a lot of difficulties in applications where sequences of events have different lengths, such as controlling the growth of plants, since the number of all possible combinations of events is intractable. Therefore, the learning method should be able to orient the search of relevant events so as to give more chances to those combinations that are more likely to be relevant according to the experience collected so far. Moreover, provided that the number of experiences is small with respect to the total number of possible combinations, the learning method should be able to accurately evaluate the relevance of a combination using only few experiences.

The following sections present how the relevance of a combination of events is evaluated and how new combinations are generated.

4.1. Rule Evaluation

The evaluation of a combination of events in the precondition part of a cause-effect rule is carried out using the conditional probability

$$\mathscr{P} = \Pr(E \,|\, P, A), \tag{19}$$

where E is the effect part of the rule, P is the precondition part, formed by the tested combination of past events, and A is the sequence of actions of the rule. This probability could be calculated using many different methods such as the *m*-estimate or a pure frequency approach (Furnkranz and Flach, 2003). In our case, we will use the estimate proposed in (Agostini et al., 2011), the *density*-estimate, since it compensates the lack of experience by considering in the estimate not only the experienced instances but also the inexperienced ones, avoiding biased premature estimations when few examples are collected. The formula of the *density*-estimate is,

$$\mathscr{P} = \frac{n_+ + \hat{n}_\emptyset \, c}{n_+ + n_- + \hat{n}_\emptyset}, \tag{20}$$

where n_+ is the number of *positive* instances, n_- is the number of *negative* instances, c is an *a priori* probability, usually set to $c = 1/2$, and $\hat{n}_\emptyset$ is an estimation of the number of all possible instances covered by the rule that are still pending to be experienced. The parameter $\hat{n}_\emptyset$ represents the uncertainty in the probability estimation: the larger its value the larger the uncertainty.

In practice, the value of $\hat{n}_\emptyset$ is defined as $\hat{n}_\emptyset = n_T - n_+ - n_-$, where n_T is the estimated number of instances which are required to be experienced to consider the probability estimation as confident. The parameter n_T is defined depending on the total number of instances covered by the rule. If this number is relatively small we could set n_T to this number, assuming all the instances can be experienced in a reasonable amount of time. This would provide a fully confident estimation when all the covered instances are experienced, i.e. when $\hat{n}_\emptyset = n_T - n_+ - n_- = 0$. On the contrary, if the total number of covered instances is large, it may be impractical, or even infeasible, to experience all the covered instances. In this case, the parameter n_T should be defined by the user, ideally by an expert user knowing the application at hand, depending on how many instances are necessary to trust the probability estimation. In the latter case, if $n_T < n_+ + n_-$, then $\hat{n}_\emptyset$ is set to 0 and the estimate (20) becomes the traditional frequency estimate.

Note that the *density*-estimate uses the density of samples in the region of the sequence space covered by the rule. The lower the density of samples, the higher the uncertainty, and the lower the variation of the probability in the estimation. Note that a high uncertainty exists when the number of experienced instances is much smaller than n_T, i.e. when $n_T >> n_+ + n_-$. In this case, according to Eq. (20), if we have two rules with few experienced instances, their probability estimates will be similar and close to the predefined prior probability c, which prevents a biased probability estimation when few experiences are provided.

4.2. Rule Generation

The strategy for rule generation is in charge of finding those combinations of events that, considered in the precondition part of a cause-effect rule, will produce the effects coded in the rule with high probability (19). There are two instances in which a new rule is generated. The first one takes place when the executed sequence of actions was instructed by the teacher. The second instance of rule generation is when the execution of a rule does not produce the expected effect.

4.3. Rule Generation from Instruction

After the execution of a sequence of actions instructed by the human expert, a new rule is generated from the observed changes. The action part of the new rule

consists of the instructed sequence of actions, where the first action is considered to take place at time-step $t = 0$. The precondition and effect parts are created using the state descriptors that have changed with the sequence of actions. The precondition part is formed by the initial values of the changed descriptors, all considered to take place at time-step $t = 0$. The effect codes the evolution of the changed descriptors with each action in the sequence.

To illustrate how this is carried out we use the example sequence presented in Table 1, where the changed descriptors are extracted from a sequence of states observed during the execution of a two actions sequence, say $\{a_2^0, a_1^1\}$. The parts of the rule generated from the observed changes would be

$$\begin{aligned} P_1 &= \{d_{1,3}^0, d_{3,1}^0\}, \\ A_1 &= \{a_2^0, a_1^1\}, \\ E_1 &= \{d_{1,3}^1, d_{3,2}^1, d_{1,4}^2, d_{3,2}^2\}. \end{aligned} \tag{21}$$

All the rules generated from instruction are coded as planning operators using the strategy explained in Sec. 2.5 and become immediately available for planning.

Table 1: Example of changed state descriptors

State Sequence	Changed Descriptors
$s^0 = \{d_{1,3}^0, d_{2,2}^0, d_{3,1}^0, d_{4,4}^0\}$	$\{d_{1,3}^0, d_{3,1}^0\}$
$s^1 = \{d_{1,3}^1, d_{2,2}^1, d_{3,2}^1, d_{4,4}^1\}$	$\{d_{1,3}^1, d_{3,2}^1\}$
$s^2 = \{d_{1,4}^2, d_{2,2}^2, d_{3,2}^2, d_{4,4}^2\}$	$\{d_{1,4}^2, d_{3,2}^2\}$

4.4. Rule Generation from Unexpected Effects

The execution of a rule having missing relevant events in its precondition part may produce changes different from the ones coded in the effect part. Such unexpected effects trigger the mechanism for refining the failing rule.

An unexpected effect occurs if the expected sequence of changes is different from the observed one, which is determined using the previously introduced metric (4) as

$$\mathrm{diff}(E_{exe}^t, S^t) > \mathrm{thr}_{\mathrm{eff}}, \tag{22}$$

where S^t is the sequence of states observed from the execution of the first action in the rule (i.e. at time-step $t = 1$) up to time-step t, E^t_{exe} is the sequence of expected effects up to time-step t, and thr_{eff} is the threshold to consider the observed sequence as an unexpected effect. This threshold permits considering deviations from the expected evolution produced, for instance, by hidden variables or by the stochastic nature of the environments. In this case, the threshold thr_{eff} should be set by the expert user larger than, and close to, 0 depending on what deviation is expected from the stochastic components. In deterministic fully observable environments, instead, this threshold should be set to 0 since any deviation from the expected evolution of events is produced by missing past events in the precondition part of the rule.

If an unexpected effect occurs, many specializations of the failing rule are generated and the one with highest probability (19) is used to code a new PO that replaces the failing one. To quickly find potentially relevant combinations of events, our strategy consists in storing in memory the most accurate rules for each of the coded changes. When an unexpected effect occurs, all the rules coding the same changes of the failing one are brought together in the set

$$\mathscr{R}_{exe} = \{r | A = A_{exe}, E = E_{exe}\}. \tag{23}$$

Then, the set $\mathscr{R}_{exe}$ is expanded as

$$\mathscr{R}_{exe'} = \mathscr{R}_{exe} \cup \mathscr{R}_{basic}. \tag{24}$$

The aim of $\mathscr{R}_{basic}$ is to ensure that all the possible combinations of events have a chance to occur. To this end, possible past events should be available individually for the combination among them and with the ones in $\mathscr{R}_{exe}$. This is implemented by including in $\mathscr{R}_{basic}$ rules having in their precondition part a single event in addition to those changing with the action. The added events are extracted from the positive instances of the failing rule to prevent evaluating events with no evidence to produce the expected changes. To gradually explore the past, we take events between the reference time $t = 0$ and $t = t_{min} - 1$, where t_{min} is the lower time-step in the precondition part of the failing rule, P_{exe}.

After generating the set $\mathscr{R}_{exe'}$, new combinations of events are generated by randomly selecting n rules from $\mathscr{R}_{exe'}$ according to their probability (19) and combining them into a new set

$$\mathscr{R}_{comb} = \{r_k | P_k = \{P_i \cup P_j\}, A_k = A_{exe}, E_k = E_{exe}\}, \tag{25}$$

where P_i and P_j, $i \neq j$, are the precondition parts of any two rules of the n selected. The random selection according to the rule probability favours the combination of events presenting more evidence to be accurate while allowing also for combination of events that may not produce a high probability when considered separately but that may actually lead to a high probability when combined. Note that we could combine all the rules in $\mathscr{R}_{exe'}$ instead of n of them. This would actually increase the chances of producing relevant combinations of descriptors but at the expense of a significantly higher computational cost.

Finally, we merge the previously generated sets into the candidate set of rules,

$$\mathscr{R}_{cand} = \mathscr{R}_{exe'} \cup \mathscr{R}_{comb}, \tag{26}$$

from which the m most accurate, $\mathscr{R}_m \subset \mathscr{R}_{cand}$, are stored in memory for future refinements. In particular, the one with highest probability, called the *winner* rule r_w, is selected to replace the failing rule, with

$$w = \underset{j \in \mathbf{i}_{cand}}{\operatorname{argmax}} \mathscr{P}_j, \tag{27}$$

where $\mathbf{i}_{cand}$ is the set of indexes for the rules in $\mathscr{R}_{cand}$, and $\mathscr{P}_j$ is the probability for rule $r_j \in \mathscr{R}_{cand}$.

4.5. Rule Elimination

To limit the proliferation of rules we eliminate those that are *redundant* with other rules in the system, i.e. which are coding similar subspaces. The redundancy between two rules r_i and r_j is calculated as

$$\mathrm{red}_{i,j} = 1 - \mathrm{diff}(q_i, q_j), \tag{28}$$

where $q_i = \{P_i \cup A_i \cup E_i\}$ and $q_j = \{P_j \cup A_j \cup E_j\}$. Two rules are considered as redundant when

$$\mathrm{red}_{i,j} \geq \mathrm{thr}_{\mathrm{red}}, \tag{29}$$

where $\mathrm{thr}_{\mathrm{red}}$ is the redundancy threshold. In case two rules are redundant we eliminate the one matching less experienced instances. The threshold $\mathrm{thr}_{\mathrm{red}}$ is usually set close to 1, or to 1, depending on the intrinsic characteristics of the environment. For instance, in deterministic fully observable environments, this threshold should be set to 1 since two rules coding a similar, but not equal, evolution of events would be actually applicable to two different states. In this case, the two

rules are redundant if they cover the same subspace of sequences, i.e. if the distance between the coded subspaces is 0. If that is so, the most specific rule, which would match less instances, is eliminated, contributing to the generalization of the system. On the contrary, a slight difference between rules may be acceptable (i.e. a redundancy threshold close to 1) if, for instance, the perception module (Fig. 1) generates some state descriptors with noisy values that are wrongly considered as part of the changes produced by the execution of a treatment. In this case, a redundancy threshold smaller than 1 would help to get rid of these rules by eliminating the noisy ones since, in the long run, they will cover less experienced instances.

At a predefined number of iterations it_{elim}, we eliminate all the rules that fulfil the elimination criterion.

4.6. The Learning Algorithm

The processes for learning rules coding weakly correlated cause-effects are summarized in Algorithm 2. These processes replace the LEARNER function in the algorithm of the decision-making framework (Algorithm 1).

Algorithm 2 $\mathscr{R} \leftarrow \text{LEARNER}(\mathscr{R}, \hat{s}^t)$

```
Update probabilities 𝒫 of rules in ℛ using instance ŝ^t
Get action a^{t-1} from ŝ^t
if a^{t-1} belongs to an instructed sequence of actions then
    if a^{t-1} is the last action in the sequence then
        Generate r_inst {rule generated from instruction (Sec. 4.3)}
        ℛ ← {ℛ, r_inst}
        Generate a PO from r_inst
    end if
else
    Get executed rule r_exe from ℛ
    Get sequence of states S^t from ŝ^t
    if diff(E^t_exe, S^t) > thr_eff then
        Generate ℛ_cand {candidates rules for r_exe}
        Extract most accurate rules ℛ_m from ℛ_cand
        ℛ ← {ℛ ∪ ℛ_m}
        w = argmax_{j∈i_cand} 𝒫_j {get index of the winner rule}
        Generate a PO from r_w to replace the failing one
    end if
end if
if (t modulo it_elim)=1 then
    Eliminate rules fulfilling the elimination criterion {Sec. 4.5}
end if
```

5. Application Example: Controlling the Growth of Plants

A typical application where weak correlations are involved is the gardening application of controlling the growth of plants. In this application, the correlation

between the actions taken on the plants and their effects take place with variable delays and strength, making it necessary to analyse the long-term evolution of the plants to detect these weak correlations.

Since using real plants for the experiments would take an infeasible amount of time, we adopt the usual approach in botany and agriculture of analysing the plant growth processes using simulated scenarios (Fourcaud et al., 2008; Hackett, 1973; Paine et al., 2012; Kang et al., 2012). This allows generating data in a reasonable amount of time that could otherwise take months, even years, to obtain from real plant experiments. The details of the used simulated scenario are presented in Appendix A.

5.1. Problem Definition

In order to apply our cognitive system to the task of controlling plant growth we need to define the state and action spaces introduced in Section 2. This definition was carried out with the help of an expert gardener from the project GARNICS (2010-2013) so as to make sure that the problem definition is the proper one to perform an efficient execution of the task.

The state space is defined by a set of 12 state descriptors representing different attributes of the plant and environmental conditions. The complete set of descriptors with their corresponding values is presented in Table 2. The values of the descriptors are extracted from the plant simulator signals, most of them following a discretization process. In the table, we specify the intervals employed in the discretization process, when applicable. For the definition of the number of yellow leaves, we say that a leaf is yellow when the color of the leaf $C(t)$ goes below 0.9 (see Eq. A.3 in Appendix A). The descriptor *plant height* corresponds to the height of the stem.

The action space consists of 6 possible treatments, a_k^t, $k = 1,..,6$, enumerated in Table 3. To define each treatment we use sequences of 24 atomic actions applied in a per hour basis. An atomic action consists of a dose of water, a dose of nutrients, and a percentage of light intensity variation applied to the plant (see Table 4). In this way, an action for planning is defined as a sequence of 24 atomic actions belonging to a treatment (i.e. the time interval t involves multiples of 24 hours). In practice, some of the 24 atomic actions consist just in 0 values. For the nutrient income we use a nutrient-water solution. Therefore, every nutrient intake is also considered a water intake. In Table 3 we reference the tables with the atomic actions corresponding to each of the treatments.

Table 2: State space definition.

i	Descriptor (d_i)	Values ($d_{i,j}$)
1	Number of leaves	{0,1,2,3,4,5,6,7,8,9,10,>10}
2	Number of green leaves	{0,1,2,3,4,5,6,7,8,9,10,>10}
3	Number of yellow leaves	{0,1,2,3,4,5,6,7,8,9,10,>10}
4	Ratio green/total leaves	{<0.2,[0.2,0.4),[0.4,0.6),[0.6,0.8),>0.8}
5	Mean size of leaves (cm^2)	{[0,1),[1,2),[2,3),[3,4),>4}
6	Size of smallest leaf (cm^2)	{[0,0.5),[0.5,1),[1,1.5),[1.5,2),>2}
7	Size of largest leaf (cm^2)	{[0,2.5),[2.5,3.5),[3.5,4.5),[4.5,5.5),>5.5}
8	Plant height (cm)	{<1, [1,2), [2,3), [3,4],>4 }
9	Mean growth rate of leaves (%)	{<-100,[-100,-80),[-80,-60),[-60,-40), [-40,-20),[-20,0),0,(0,20),[20,40), [40,60),[60,80),[80,100],>100 }
10	Largest leaf growth rate (%)	{0,(0,20),[20,40),[40,60),[60,80),[80,100],>100}
11	Humidity (%)	{0,10,20,30,40,50,60,70,80,90,100}
12	Temperature ($^\circ C$)	{<15,15-20,21,22,23,24,25-30,>30}

5.2. Teacher Interfaces

To provide the human expert with the information required to instruct the treatments to apply when the planner is not able to make a decision, we developed two main graphical user interfaces (GUIs): a GUI for monitoring the plant and system state and a GUI for instructing the treatments.

Fig. 2 presents a snapshot of the GUI for the monitoring of the plant and the system. The GUI shows the general status of a simplified diagram of the DMF, at the upper left corner, where the currently activated module is marked in red (the learner module in the example). The goal specification and the command buttons to run and to stop the system (and the simulation) are placed in the upper middle side of the GUI (for an explanation about the goal specification, please refer to Sec. 6). The upper right side of the GUI shows the evolution of the size

Table 3: Action space definition.

k	Treatment (a_k)
1	Extra mild watering (Table 5)
2	Mild watering (Table 6)
3	Mild nutrient (Table 7)
4	Strong watering (Table 8)
5	Strong nutrient (Table 9)
6	Strong watering and nutrient (Table 10)

and number of leaves, the color of the leaves, and the concentration of water and nutrients in the soil.

The GUI also shows the rule that is currently executed, with its precondition, action, and effect parts, and the observed state-action sequence, both in the lower half of the GUI. To deploy the rule and the state-action sequence we use a time window of plus-minus 48 hours. The state descriptors observed at specific time-steps in this time window are shown in tables painted in blue, while the actions are shown in red. To reference state descriptors, the GUI shows a short name from the name of descriptors enumerated in Table 2, e.g. 'n Leaves' for the number of leaves. For each treatment (see Table 3), each of the atomic actions is specified using the notation A(h), where A takes values 'W' for a dose of water and 'N' for a dose of nutrient, and h indicates the hour at which this dose is provided to the soil, with respect to the beginning of the treatment. Note that, for the rule specification, only the values of the state descriptors involved in the precondition and effect parts are displayed.

Every time the planner cannot make a decision due to missing POs the system deploys the GUI shown in Fig. 3 for the input of the treatments to execute. The GUI lets the teacher select and edit a single or a combination of treatments from a list at the left hand side of the GUI, or create a new combination of treatments, if required. For instance, in the snapshot of the GUI, the treatments enumerated in Table 3 are specified as: xmW (extra mild watering), mW (mild watering), mN (mild nutrient), sW (strong watering), sN (strong nutrient), and sWN (strong watering and nutrient). The rest of the elements of the treatment list consists of combinations of these basic treatments. For instance, the combination selected in the example figure is sW_sN, which corresponds to applying, first, a treatment of strong watering and, second, one of strong nutrient.

The edition/creation of a treatment is carried out through two editable tables at the right hand side of the GUI. The upper table (named 'Actions') specifies each of the atomic actions corresponding to the selected treatments. Each column corresponds to an atomic action, where the day and hour at which the atomic

Table 4: Atomic actions.

Action Names	Action Values
Dose of water (ml)	{0,1,2,3,4,5,6,7,8,9,10}
Dose of nutrient (ml)	{0,1,2,3,4,5,6,7,8,9,10}
Light variation (%)	{-50','-20','-10','0','10','20','50'}

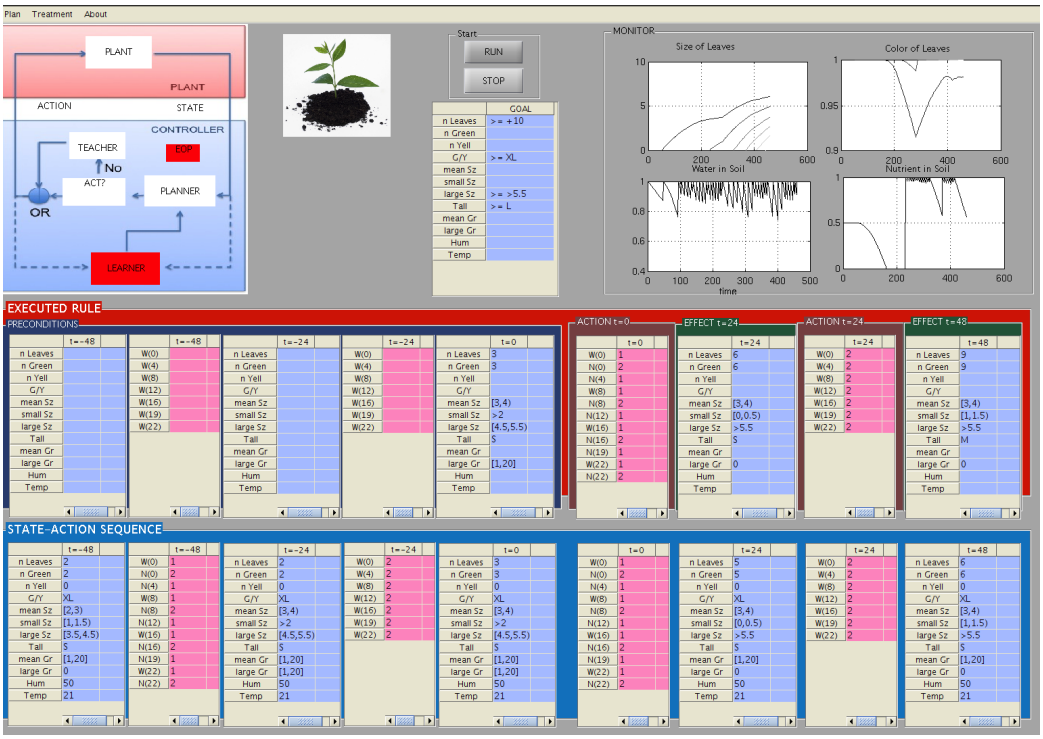


Figure 2: Snapshot of the GUI to monitor the plant and the system.

action should be executed is also shown. If the teacher inputs a new sequence of atomic actions, the system splits this sequence into sequences of 24 hours each and checks whether these sequences already correspond to any of the existing treatments. If a sequence is not already considered as a treatment, the system creates a new treatment from this sequence and adds it to the list.

The lower table at the right-hand side of the instruction GUI (named 'Attributes') permits specifying the expected evolution of each of the state descrip-

Table 5: Extra mild watering.

Atomic action	$t+0$
Water	1
Nutrient	0
Light	0

tors for the selected treatments. If the teacher provides any input to this table, the system automatically creates a cause-effect rule using the specified descriptors to set the precondition and effect parts, and the treatments in the action table to set the action part. All the descriptors that should be observed before the beginning of the treatment are used to create the precondition part of the rule, and all those after the beginning of the treatment, will form the effect part. If the specified evolution of descriptors does not match the observed one when executing the instructed treatments, the system automatically corrects those descriptors that are not matching the teacher specification. In this work, the teacher does not provide any input to this table so as to let the learning method autonomously generate the precondition and effect parts of the rules from the changes observed after the instructed treatments execution (Sec. 4.3).

The selected treatment, or sequence of treatments, is sent to the system for the execution after pressing the 'Send Instruction' button at the lower left side of the instruction GUI.

Finally, we show, in Fig. 4(a), the GUI deployed to input the goal at the beginning of the system running, after the user presses the button 'Run' in the monitoring GUI (Fig. 2). To input the value of each descriptor in the goal specification, the system displays a GUI with all the possible values for the descriptor selected in the goal GUI. For instance, Fig. 4(b) presents an example of a GUI listing the possible values for the descriptor of the size of the largest leaf.

Table 6: Mild watering.

Atomic action	$t+0$	$t+5$	$t+10$	$t+15$	$t+20$
Water	1	1	1	1	1
Nutrient	0	0	0	0	0
Light	0	0	0	0	0

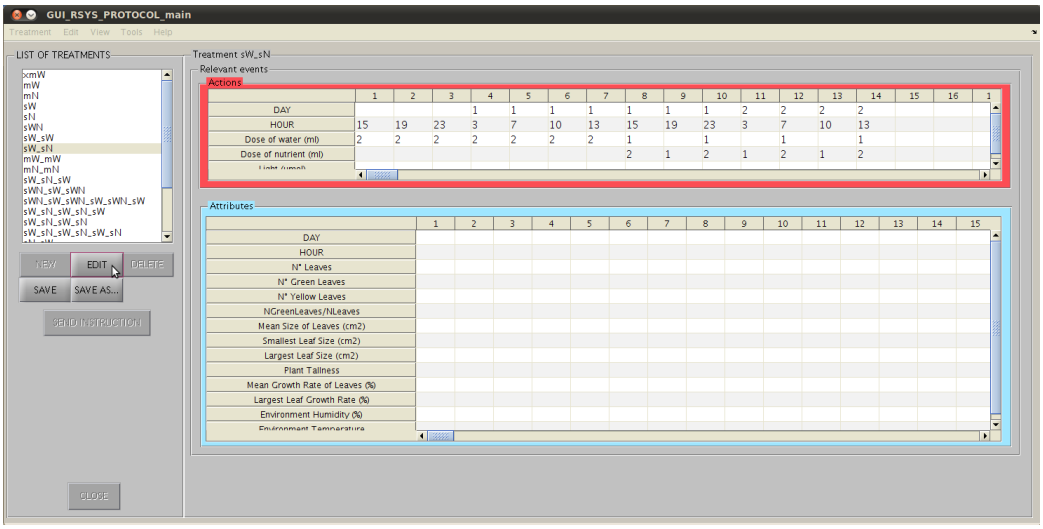


Figure 3: Snapshot of the GUI to instruct treatments to execute.

Table 7: Mild nutrient.

Atomic action	$t+0$	$t+5$	$t+10$	$t+15$	$t+20$
Water	1	0	1	0	1
Nutrient	1	1	1	1	1
Light	0	0	0	0	0

Table 8: Strong watering.

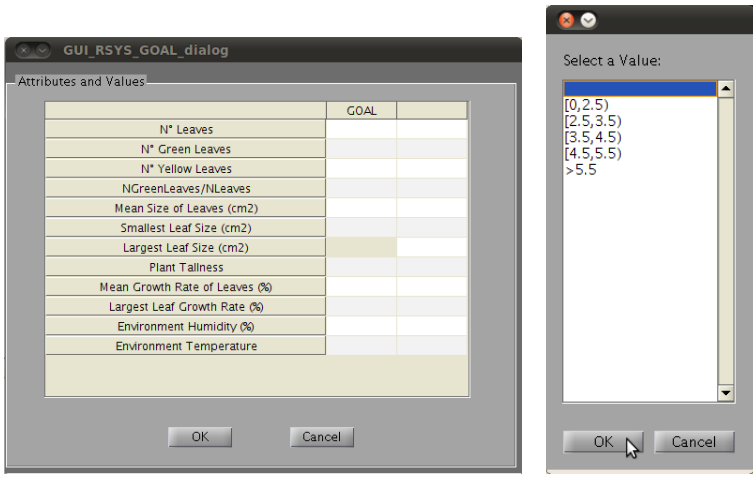
Atomic action	$t+0$	$t+4$	$t+8$	$t+12$	$t+16$	$t+20$	$t+23$
Water	2	2	2	2	2	2	2
Nutrient	0	0	0	0	0	0	0
Light	0	0	0	0	0	0	0

Table 9: Strong nutrient.

Atomic action	$t+0$	$t+4$	$t+8$	$t+12$	$t+16$	$t+20$	$t+23$
Water	1	0	1	0	1	0	1
Nutrient	2	1	2	1	2	1	2
Light	0	0	0	0	0	0	0

Table 10: Strong watering and nutrient.

Atomic action	$t+0$	$t+4$	$t+8$	$t+12$	$t+16$	$t+20$	$t+23$
Water	2	2	2	2	2	2	2
Nutrient	2	1	2	1	2	1	2
Light	0	0	0	0	0	0	0



(a) goal GUI (b) values GUI

Figure 4: Snapshots of the GUI to specify the goal and a descriptor value.

6. Experiments

The experiments are carried out using a set of planning problems with initial situations obtained by randomly selecting and applying two planning actions from Table 3 for the first 48 hours of treatments. The goal is defined by four descriptors representing the size and healthiness of the plant attributes. To specify the desired size of the plant we use: number of leaves > 10, largest leaf size > 5.5, and height of the plant > 4. The healthiness of the plant is considered by a ratio between the number of green leaves versus the total number of leaves larger than 0.8.

For the experiments we use the same planner as in Agostini et al. (2011), the PKS planner (Petrick and Bacchus, 2002). In the process, no rules are eliminated to have a better idea of the rate of rule generation. The rule database is initially empty. The number of instances to consider the estimation as confident in the *density*-estimate formula is set to $n_T = 50$. We set the threshold for detecting unexpected evolutions of the plant to $\text{thr}_{\text{eff}} = 0$.

6.1. Reference Performance: The OBSERVER Method

In order to evaluate performance, we compare the results obtained with our learning approach with those obtained by using one of the most well-known approaches for online learning of planning operators, the OBSERVER method (Wang, 1995, 1996), adapted to the learning of weak correlations.

In this method, learning and planning are interleaved so as to acquire planning operators in a learning by doing approach. A human expert provides solution traces offline, which are used by OBSERVER to generate an initial set of rules that are later refined from experience using a simulated environment. The refinement from experience takes place using two sets of samples. The set of samples where a rule is *successfully executed* (SUC) and the set of samples where the rule *fails-to-execute* (FAIL). The OBSERVER method is an adaptation of the Version Space method (Mitchell, 1997), which, for each rule, keeps and refines the most specific and the most general representation of its precondition part. The most general precondition representation is used by a modified PRODIGY planner that is able to perform a plan repair, using the most specific representation, every time a plan execution fails.

To carry out the comparison, we adapted the OBSERVER method to our decision-making framework for the learning of rules coding weakly correlated cause-effects. As in our case, rules are generated online, after each action instruction, and not offline from solution traces provided by an expert. The most general representation is initialized to the observed changes in the sequence in the

same way as we do in our method (see Section 4.3). The most specific precondition representation is initialized with all the descriptors and actions of the past 5 days. This number of days corresponds to the maximum delay needed to accurately predict the evolution of plants from different initial situations observed in the experiments.

Once a rule is generated from an instruction, the most specific and most general representations are refined using the mechanisms described in Wang (1995). As in our case, these refinements are carried out on the failing rule if an unexpected effect occurs (see Alg. 2). The refinement takes place using the positive and negative instances to create the sets SUC and FAIL, respectively. In general terms, the most specific representation is refined from positive instances in SUC. If OBSERVER detects that an event in the most specific precondition representation is not present in any positive instance, then it deletes the event from this representation.

On the other hand, the most general precondition representation is refined in two ways: by comparing the most specific representation with a negative instance and by comparing positive and negative instances. If only a single event in the most specific representation is not present in a negative instance while the rest of them are, then this event is added to the most general representation. In a similar way, if the difference between any positive and any negative instance is in only one event, then this event is added to the most general precondition representation. In this way, the most general representation is only refined with events proving high relevancy.

Finally, since we use the PKS planner in our DMF, which does not perform a plan repair when a plan execution fails, we only use the most general representations in the generation of planning operators.

6.2. Results

Figure 5 presents the results of the experiments for our approach (blue solid lines) and for the OBSERVER method (red dashed lines). The figure shows the accumulated number of teacher instructions during learning vs. the total number of actions (iterations) (Fig. 5 A), the accumulated number of unexpected effects (Fig. 5 B), the number of rules generated during learning (Fig. 5 C), and the ratio between the number of plans completed successfully, i.e. without instructions or unexpected effects, and the total number of plans (Fig. 5 D).

As we can see, our learning method generates rules that rapidly improve the decision-making performance, permitting the system to successfully execute 70 % of the plans after about 100 planning problem executions and 95 % of the plans

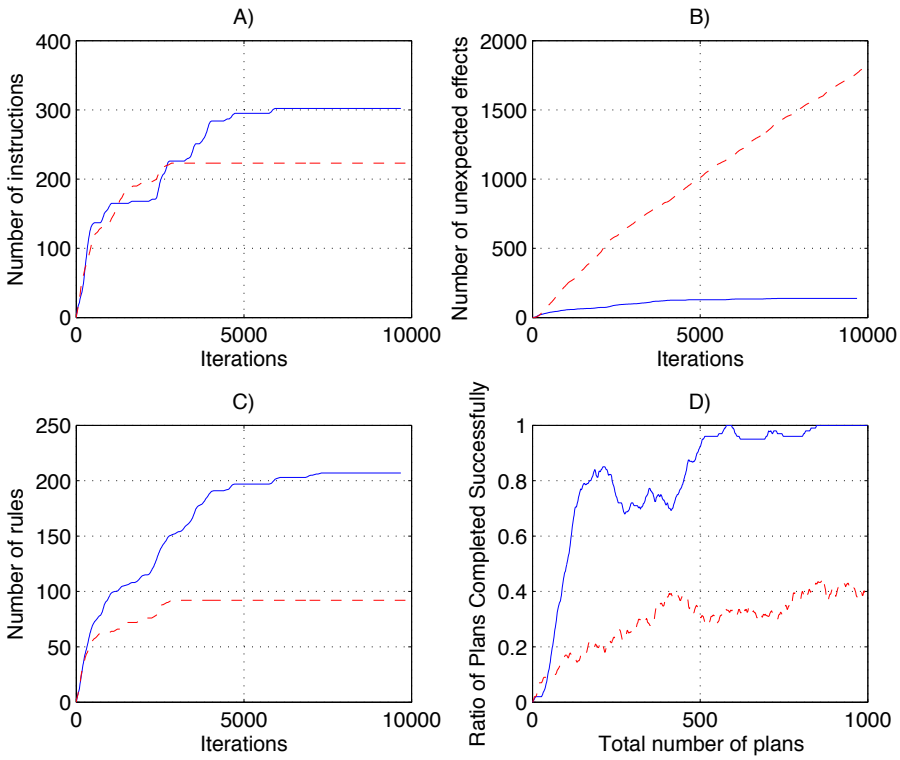


Figure 5: Results obtained with our learning approach are plotted in continuous blue lines while the ones obtained with the OBSERVER method are shown in dashed red lines. A) Accumulated number of teacher interventions during learning. B) Accumulated number of unexpected effects during learning. C) Accumulated number of rules generated during learning. D) Ratio of plans completed successfully (without teacher instructions or unexpected effects).

after about 500 executions. Rule generation and refinement are very intense at the beginning of the learning. The teacher instructs several actions relevant for the task, mainly at the beginning of the learning when no rules are available to the planner. The execution of the instructed actions leads to the generation of rules coding useful treatments to produce the needed changes to achieve the goal. Then, these rules are progressively refined by the learning method, which is able to find relevant events in the past history of the plant that actually allows the coded changes to occur. The refinement process takes place until no unexpected plant evolution occurs, permitting a successful execution of 100 % of the plans at later

stages of learning.

To provide a concrete example of the learning capabilities of our approach, Table 11 shows a rule generated from instruction, coding only the instructed actions and the changes observed after their execution (see Sec. 4.3). We use a table representation for clarity in the notation, where the precondition, action, and effect parts of the rule are placed in the first, second, and third columns of the table, respectively. We explicitly indicate a short name and the value of each descriptor involved in the rule (see Table 2), as well as the time-step, in multiples of 24 hours, at which they are observed. The actions coded in the rule are also explicitly referenced with the treatment applied at each time-step (see Table 3).

The example rule was generated after the execution of two instructed treatments of strong nutrient (Table 9), applied at intervals of 24 hours: the first one started at the beginning of the execution of the rule, i.e. at 0 hours, and the second one one time-step later, i.e. 24 hours later. These treatments define the action part of the rule, as shown in the second column of Table 11. The execution of the two instructed treatments produced the changes in the descriptors: number of leaves (n leaves), number of green leaves (n green leaves), mean size of leaves (Mean Sz), size of the smallest leaf (Small Sz), size of the largest leaf (Large Sz), plant height, and mean growth of leaves (Mean Gr). In the first column of the table we show the initial values of these descriptors, which defines the precondition part of the rule. In the third column of the table, we first show the values of the descriptors after the first 24 hours from the beginning of the rule execution, i.e. once the first treatment had ended, and then, the values observed at the end of the execution of the two treatments, i.e. 48 hours after the beginning of the rule execution. We can see, for instance, that the number of leaves has changed from 1, at the beginning of the execution, i.e. at $t = 0$, to 2, at the first time-step of 24 hours, and to 4, 48 hours later than $t = 0$.

Due to the delayed effects of actions, considering only the initial values of the changed descriptors as a precondition may not be enough to guarantee that the encoded evolution of the plant will occur. This is actually the case for the example rule, which is reflected in the number of positive and negative instances associated to the rule presented in Table 15. The rule has 111 positive instances, i.e. 111 state-action sequences where the observed changes occurred (see Sec. 2.4), but a larger number of 189 negative instances, where the coded changes were not obtained even though the precondition and action parts of the rule matches the sequences.

For an accurate prediction, the learning approach needed to find which other events in the past history of the plant should also be observed in order to obtain

the coded sequence of changes. For the case in point, our method was able to find out, from all the possible past events, that the relevant ones for an accurate prediction are the last two actions applied to the plant, i.e. two treatments of mild watering (Table 6) 24 and 48 hours before the beginning of the rule execution, as well as some past values of the descriptors that had changed with the actions: 0 green leaves 48 hours before the rule execution (n green leaves=0 at -48 h), size of the smallest leaf in the range [0,0.5) cm^2 24 hours before the rule execution (Small Sz $\in$ [0,0.5) at -24 h), and plant height lower than 1 *cm* 48 hours before the rule execution (Plant height $<$ 1 at -48 h). The precondition part of the rule after this refinement is shown in Table 14. Tables 12 and 13 present two intermediate stages of the refinement of the precondition part so as to exemplify the refinement process. The statistics associated to all these refinements are shown in Table 15. Note that the most accurate refinement of Table 14 has the highest number of 57 positive instances, and no negative ones.

Table 11: Rule refinement example: first rule generated from observed changes.

Preconditions	Actions	Effects
n leaves = 1 at 0 h	Strong nutrient at 0 h	n leaves = 2 at 24 h
n green leaves= 1 at 0 h		n green leaves = 2 at 24 h
Mean Sz $\in$ [2,3) at 0 h		Mean Sz $\in$ [2,3) at 24 h
Small Sz $>$ 2 at 0 h		Small Sz $\in$ [1,1.5) at 24 h
Large Sz $\in$ [0,2.5) at 0 h		Large Sz $\in$ [3.5,4.5) at 24 h
Plant height $<$ 1 at 0 h		Plant height = [1,2) at 24 h
Mean Gr $\in$ (0,20] at 0 h		Mean Gr $\in$ (0,20] at 24 h
	Strong nutrient at 24 h	n leaves = 4 at 48 h
		n green leaves = 4 at 48 h
		Mean Sz $\in$ [3,4) at 48 h
		Small Sz $\in$ [0,0.5) at 48 h
		Large Sz $\in$ [4.5,5.5) at 48 h
		Plant height = [1,2) at 48 h
		Mean Gr $\in$ [-40,-20) at 48 h

Contrarily to our approach, the OBSERVER method does not perform well in the selected application. Even though it is indeed able to find some relevant events in the past, it is not able to completely refine all the rules, only reaching in the long term a ratio of 40 % of the plans successfully executed (see Fig. 5 D). One reason for the poor performance might be that it does not explore past events that may produce the effects coded in a disjunctive form. Having events with disjunc-

Table 12: Precondition part of the example rule after the first refinement

n leaves = 1 at 0 h
n green leaves= 0 at -48 h
n green leaves= 1 at 0 h
Mean Sz $\in$ [2,3) at 0 h
Small Sz $>$ 2 at 0 h
Large Sz $\in$ [0,2.5) at 0 h
Plant height $<$ 1 at 0 h
Mean Gr $\in$ (0,20] at 0
Action mild watering at -24

Table 13: Precondition part of the example rule after the second refinement

n leaves = 1 at 0 h
n green leaves= 0 at -48 h
n green leaves= 1 at 0 h
Mean Sz $\in$ [2,3) at 0 h
Small Sz $\in$ [0,0.5) at -24 h
Small Sz $>$ 2 at 0 h
Large Sz $\in$ [0,2.5) at 0 h
Plant height $<$ 1 at -48 h
Plant height $<$ 1 at 0 h
Mean Gr $\in$ (0,20] at 0
Action mild watering at -24

tive form is a frequent case in the gardening application, where plants may have similar evolutions under different combination of past events. Note that, in the OBSERVER mechanism for the most specific precondition refinement, an event is deleted if it is not present in all the positive instances (see Sec. 6.1). Hence, disjunctive events, which would appear in some but not all positive instances, would not be considered for the refinements of the most general precondition.

Our approach is instead able to consider disjunctive events, since all the events in positive instances have a chance to be evaluated, disregarding whether they appear in some or all of the positive instances. To provide an example, we use the rule presented in Table 16, generated after the teacher instruction of executing in sequence three treatments: a strong nutrient treatment (Table 9), starting at $t = 0$ hours, another strong nutrient treatment, starting at $t = 24$ hours, and a treatment

Table 14: Precondition part of the example rule after the third refinement

n leaves = 1 at 0 h
n green leaves= 0 at -48 h
n green leaves= 1 at 0 h
Mean Sz $\in$ [2,3) at 0 h
Small Sz $\in$ [0,0.5) at -24 h
Small Sz $>$ 2 at 0 h
Large Sz $\in$ [0,2.5) at 0 h
Plant height $<$ 1 at -48 h
Plant height $<$ 1 at 0 h
Mean Gr $\in$ (0,20] at 0
Action mild watering at -24
Action mild watering at -48

Table 15: Rule refinement example: statistics

Rule	n^+	n^-
Rule generated from inst. (Table 11)	111	189
First refinement (Table 12)	98	36
Second refinement (Table 13)	98	36
Third refinement (Table 14)	57	0

of strong watering (Table 8), starting at $t = 48$ hours. The descriptors changed with the execution of these treatments are: number of leaves (n leaves), number of green leaves (n green leaves), mean size of leaves (Mean Sz), size of the smallest leaf (Small Sz), size of the largest leaf (Large Sz), plant height, and growth rate of the largest leaf (Large Gr). Tables 17 and 18 present two refinements of the precondition part of the example rule carried out by our approach. These refinements involve two disjunctive events that should take place 24 hours before the rule execution: the action of mild watering (Table 17) and the action of strong watering (Table 18). Any of these actions would produce the same changes with the additional condition that the action of extra mild watering should be applied 48 hours before the rule execution. Both disjunctive refinements improve the accuracy of the original rule as shown in Table 19. In this table we can see that the rule generated from instruction has associated 241 negative instances and only 58 positive instances, while the two refinements have no negative instances, and 36 and 20 positive instances.

Table 16: Disjunctive form example: first rule generated from observed changes.

Preconditions	Actions	Effects
n leaves = 1 at 0 h	Strong nutrient at 0 h	n leaves = 2 at 24 h
n green leaves= 1 at 0 h		n green leaves = 2 at 24 h
Mean Sz $\in$ [2,3) at 0 h		Mean Sz $\in$ [2,3) at 24 h
Small Sz $>$ 2 at 0 h		Small Sz $\in$ [1,1.5) at 24 h
Large Sz $\in$ [0,2.5) at 0 h		Large Sz $\in$ [3.5,4.5) at 24 h
Plant height $<$ 1 at 0 h		Plant height $\in$ [1,2) at 24 h
Large Gr $\in$ (0,20] at 0 h		Large Gr $\in$ (0,20] at 24 h
	Strong nutrient at 24 h	n leaves = 3 at 48 h
		n green leaves = 3 at 48 h
		Mean Sz $\in$ [3,4) at 48 h
		Small Sz $>$ 2 at 48 h
		Large Sz $\in$ [4.5,5.5) at 48 h
		Plant height $\in$ [1,2) at 48 h
		Large Gr $\in$ (0,20] at 48 h
	Strong watering at 48 h	n leaves = 6 at 72 h
		n green leaves = 6 at 72 h
		Mean Sz $\in$ [3,4) at 72 h
		Small Sz $\in$ [1,1.5) at 72 h
		Large Sz $>$ 5.5 at 72 h
		Plant height $\in$ [1,2) at 72 h
		Large Gr = 0 at 72 h

Table 17: Disjunctive form example 1. Event: action mild watering at -24 h.

n leaves = 1 at 0 h
n green leaves= 1 at 0 h
Mean Sz $\in$ [2,3) at 0 h
Small Sz $>$ 2 at 0 h
Large Sz $\in$ [0,2.5) at 0 h
Plant height $<$ 1 at 0 h
Large Gr $\in$ (0,20] at 0
Action extra mild watering at -48
Action mild watering at -24

Another reason for the poor performance of OBSERVER is that the refinement of the most general precondition representation only takes place when there is

Table 18: Disjunctive form example 2. Event: action strong watering at -24 h.

n leaves = 1 at 0 h
n green leaves= 1 at 0 h
Mean Sz $\in$ [2,3) at 0 h
Small Sz $>$ 2 at 0 h
Large Sz $\in$ [0,2.5) at 0 h
Plant height $<$ 1 at 0 h
Large Gr $\in$ (0,20] at 0
Action extra mild watering at -48
Action strong watering at -24

Table 19: Disjunctive form example: statistics

Rule	n^+	n^-
Rule generated from inst. (Table 16)	58	241
Disjuntive form example 1 (Table 17)	36	0
Disjuntive form example 2 (Table 18)	20	0

only one different event between positive and negative instances or between the most specific precondition and negative instances (see Sec. 6.1). Even though this strategy permits finding descriptors with high relevancy, the situation in which this is actually happening is unusual in the application of controlling plant growth. This is so since positive and negative instances consider the evolution of plants along several days in the past, and finding two instances in which the evolution only varies in one event is rare. In this way, there are relevant events that cannot be identified by this method and some rules cannot complete their refinements, which causes unexpected effects to never cease (see Fig. 5 B). In contrast, our approach considers all the events in positive instances as potentially relevant, disregarding whether they appear in negative instances or not, and tries different combinations of them for rule refinement.

We can see in Fig. 5 A) that the accumulated number of instructions for the OBSERVER method converges faster, and to a lower value, than for our approach. The reason for this relies on the refinement of rules from unexpected effects. Note that, when a rule is refined after an unexpected effect, it usually becomes more specific, i.e. the number of situations in which the rule can be applied diminishes. Eventually, some situations turn out to have no applicable rule, making the planner

fail in finding a plan in these situations and, hence, triggering action instruction. This is actually the process taking place in our approach. This process continues until all the possible situations are covered by accurate rules, the moment at which the number of instructions and unexpected effects both converge. In the case of the OBSERVER, many of the rules producing unexpected effects are never refined due to the reason explained before, keeping the same number of situations where the rule is applicable, and triggering no further action instructions.

Finally, note that the number of rules generated by our approach is much higher than the number of rules generated by OBSERVER (Fig. 5 C). One reason for this is that our approach keeps refining rules until no rules produce unexpected effects, while the OBSERVER stops this refinement much earlier. Another reason is that OBSERVER only considers two precondition representations for the coded changes, the most general and the most specific, while in our case many different alternative preconditions are evaluated [2]. If we group together all the rules generated by our approach coding the same changes, the resulting number of rules is 50, which is comparable to the number of rules generated by the OBSERVER method.

7. Conclusions

In this work we addressed the difficult problem of controlling the growth of plants using artificial intelligence techniques for planning and learning. There are two aspects that contributed to the feasibility of the approach. On the one hand, we developed a novel method to learn planning operators coding the events in the past history of the plant that are relevant to predict the future evolution of the plant under different treatments. On the other hand, we integrated the learning method in a decision-making architecture that allows learning these operators online, while the task is executed without interruptions.

The decision-making framework interleaves the learning method with a logic-based planner and a human gardener. The integration of these three components allows for a continuous improvement in the control strategy. The planner progressively improves its performance since it uses any refinement in the coded weakly correlated cause-effects immediately after it happens. On the other hand, the human gardener speeds up the learning by instructing actions that produce

[2]For a fair comparison with our approach, we count the most specific and most general precondition representations generated by the OBSERVER as two rules.

cause-effects relevant for the task at hand when the planner cannot make a decision due to missing operators. The improvement in the performance of the system is reflected in Fig. 5 D), where the ratio of plans completed successfully is zero at the very beginning of the learning, when many human instructions take place, and one at final stages.

We observe from the experiments that the events relevant to predict the effects of a sequence of actions are, in general, past values of the plant attributes that change with the actions, as well as actions applied in the past. For instance, in the example rule presented in Table 11, the final refinement done by the learning method (Table 14) considers, from all the possible past events, five events consisting of past values of three of the attributes changed with the actions and two previously applied treatments of mild watering, which allow for the evolution of the plant coded in the rule.

To assess the validity of our approach we ran several experiments in a simulated environment and compared the results with those obtained using an adaptation of OBSERVER, a well-known method for the online learning of operators for planning (Wang, 1995). The results show that our approach significantly outperforms the OBSERVER method, it being able to find the past events that permit an accurate prediction of the evolution of plants much faster. This allows the planner to be able to complete most of the plans autonomously already at early stages of the learning.

It proved hard to find a method for comparison, since no one has so far addressed the type of problem we were addressing, namely the learning of planning operators from weakly correlated cause-effects. Thus, we had to resort to the well-known OBSERVER method and adapt it to be applicable to our problem. Although this adaptation was done in the best way we could think of, the comparison may still not be entirely fair, since our approach was devised from the very beginning specifically for learning from weak correlations. For instance, in the original version of OBSERVER, the most specific precondition representation is initialized with all the descriptors from the state before the action execution. According to our state definition for planning (16), this would involve all the past events until the time at which the rule was executed, which may entail an arbitrary large number of events. Since the maximum delay needed to accurately predict the evolution of plants from different initial situations observed in the experiments is 5 days, we avoid considering for the implementation of the OBSERVER method events farther in the past to limit the size of the search space. However, this fixed number of days may compromise the performance of the OBSERVER method when compared to our approach. This is so since many rules only need to con-

sider events from up to two or three days in the past for an accurate prediction of the effects, which involves a much smaller search space. Since this information is not available in advance, we cannot restrict the search space for OBSERVER to the optimal case for each rule. Note that our approach avoids this problem by refining the precondition part in a general to specific approach, where the explorations of events in the past is carried out gradually, according to the learning needs (see Sec. 4.4). Another factor to take into account from the implementation of OBSERVER in our decision-making framework is that the samples available for learning are only collected online, while the task is executed, and not offline, from solution traces provided by an expert, as in the original implementation of OBSERVER (Wang, 1995). This may significantly restrict the number of samples available, mainly at early stages of the learning. Having less samples, it may decrease the chances of refining the failing rule due to the fact that no sample fulfils the requirements necessary for the refinement using the OBSERVER method, as explained in Sec. 6.

In the simulated evaluation we assume a fully observable deterministic environment. This implies that plants presenting the same past history would have the same evolution under the same treatments, and that all the relevant events to predict this evolution are observable and considered in the state representation. In this way, the capability of the learning approach can be evaluated precisely since any deviation from the expected evolution of the plant would indicate that some relevant events in the past history of the plant are still missing in the precondition part of the failed rule. However, in real plant scenarios, the evolution of plants may vary even if they present the same phenotype and history. This is so since not all the factors affecting the plant development can be considered in the state description, e.g. intrinsic genetic factors. Note that our approach permits considering this variability through the threshold for the generation of rules from unexpected effects (see Eq. (22)). This threshold defines the minimum difference between the coded sequence in the rule and the observed one that would trigger the rule refinement process. By setting this threshold to the size of the variation expected from the hidden variables, we would avoid refining rules that may already take into account all the relevant observable events.

There are several possible extensions of the proposed decision-making framework. One interesting extension would be to use a probabilistic planner instead of a deterministic one (as the PKS) for decision-making so as to use the probabilistic information attached to the learned rules. Another possible extension would be to provide a more informative instruction request. So far, the instruction request only informs that a plan was not found. If only one missing operator prevents the

planner to find the plan, then the teacher may instruct several actions that produce already existing operators until the missing operator is generated. To improve performance, the planner could also provide some information about the possible missing operators that originated the instruction request. This would permit the teacher to instruct actions that generate the missing operators faster.

Acknowledgements

This research was supported by the European Communitys Seventh Framework Programme FP7/2007-2013 - Challenge 2 - Cognitive Systems, Interaction, Robotics - under grant agreement No 247947 - GARNICS.

Appendix A. Plant Simulator

Many plant simulators have been developed to study the growth of plants (Fourcaud et al., 2008). These simulators are usually focused on specific aspects of the plant behaviour, e.g. how a single leaf is developed, rather than the simulation of the whole plant. In this section we propose a complete plant simulator inspired by different approaches for modelling a plant behaviour. Our simulator considers many aspects of plant behaviour, e.g. growth of leaves, growth of stem, leaf generation and elimination, variation in the color of leaves, etc. We also simulate some aspects of the behaviour of the plant environment. We would like to point out that the main purpose of this simulator is to rapidly generate sequences of events involving weak correlations so as to test our decision-making framework. To do this, we focused mostly on defining the set of equations that model all the mentioned aspects of a plant behaviour, but not on fitting and validating the model for a particular plant.

To implement the plant simulator we adopt a non-linear model assuming an asymptotic final size of the plant with a growth profile corresponding to the monomolecular approach (Paine et al., 2012). We consider each leaf individually with attributes size and color, where the size is represented with a continuous value with units cm^2, and the color is represented with continuous values ranging in [0,1], where 0 represents a completely yellow leaf (at this point already dead), and 1 represents a completely green leaf, with the highest value of healthiness.

To simulate the growth of the leaves we use a set of delay differential equations. The growth of a leaf is modeled as

$$\frac{dS(t)}{dt} = A_S(t-\tau) - k_{n_l} n_l(t) - k_S S(t), \tag{A.1}$$

where $S(t)$ represents the size of the leaf at time t, k_S is the proportionality factor that regulates the growth rate with the size of the leaf, $n_l(t)$ is the total number of leaves at time t, k_{n_l} is the proportional factor that regulates the influence of the number of leaves on the growth rate, and

$$A_S(t-\tau) = k_{W_S} W(t-\tau) + k_{N_S} N(t-\tau) + k_{L_S} L(t-\tau) + k_{T_S} T(t-\tau) + k_{M_S} M(t-\tau), \tag{A.2}$$

is the term representing the environmental influence on the size of the leaf with a delayed response of delay τ. W, N, L, T,and M are the concentrations of water and nutrient in the soil, and the environmental light (in μmol), temperature (°C), and

humidity (%), respectively, and k_{W_S}, k_{N_S}, k_{L_S}, k_{T_S}, and k_{M_S} are the corresponding gains regulating the influence of these factors on the size of the leaf.

The variation in the color of a leaf increases with water supply and nutrient incomes and decreases with the age of the plant according to

$$\frac{dC(t)}{dt} = A_C(t-\tau) - k_G G(t), \tag{A.3}$$

where $C(t)$ is the color of the plant,

$$A_C(t-\tau) = k_{W_C} W(t-\tau) + k_{N_C} N(t-\tau) + k_{L_C} L(t-\tau), \tag{A.4}$$

is the soil and environmental influence on the color variation rate, $G(t)$ is the age of the plant in hours, and k_G regulates the color with the age of the leaf: the older the leaf the larger the color variation (old leaves get yellow more easily). k_{W_C}, k_{N_C}, and k_{L_C} are the parameters regulating the influence of water, nutrient, and light in the color variation, respectively.

The growth of the stem of the plant is determined by

$$\frac{dH(t)}{dt} = A_H(t-\tau) - k_H H(t), \tag{A.5}$$

where $H(t)$ is the height of the stem and

$$A_H(t-\tau) = k_{W_H} W(t-\tau) + k_{N_H} N(t-\tau), \tag{A.6}$$

is the influence of the water and nutrient in the height variation, regulated by the parameters k_{W_H} and k_{N_H}, respectively. The term $k_S S(t)$ diminishes the growth rate of the stem according to its height in a proportion of k_H.

Appendix A.1. Leaf Generation and Elimination

The simulator generates and eliminates leaves according to the plant and environmental conditions. Leaves are generated when the size of the stem is greater than 0, and the average water and nutrient saturations are above a threshold thr_{gen}, i.e. when the inequalities $\bar{W} > thr_{gen}$ and $\bar{N} > thr_{gen}$ are fulfilled, where

$$\bar{W} = \frac{1}{t}\int_0^t W(t)dt, \tag{A.7}$$

and

$$\bar{N} = \frac{1}{t}\int_0^t N(t)dt, \tag{A.8}$$

are the average concentrations of water and nutrient along time.

A leaf is eliminated when it is *too* yellow, which is determined by the inequality $C(t) < thr_{color}$, where thr_{color} is a threshold for the color of the leaf.

Appendix A.2. Water and Nutrient in Soil Simulation

We simulate the variation of water and nutrient concentration in the soil using a set of first order differential equations (Verkroost and Wassen, 2005). The variation of the water concentration is modeled as

$$\frac{dW(t)}{dt} = k_W \Delta_W(t) - k_{\bar{S}_W} \bar{S} - k_V W(t), \tag{A.9}$$

where $W(t)$ is the water concentration, $\Delta_W(t)$ is the water income in *ml* at time t, k_W is the parameter regulating the influence of the water income,

$$\bar{S} = \frac{1}{n_l(t)} \sum_{i=1}^{n_l(t)} S_i(t), \tag{A.10}$$

is the average size of the leaves, and $k_{\bar{S}_W}$ is the parameter regulating the influence of the average size of leaves. The term $k_V W(t)$ models the evaporation of the water in the soil: the larger the amount of water in the soil the higher the evaporation rate.

The variation of the nutrient concentration in the soil is modelled as,

$$\frac{dN(t)}{dt} = k_N \Delta_N(t) - k_{\bar{S}_N} \bar{S}, \tag{A.11}$$

where $N(t)$ is the nutrient concentration, $\Delta_N(t)$ is the nutrient income in *ml* at time t, and $k_{\bar{S}_N}$ regulates the influence of the average size of leaves in the nutrient concentration. Note that, in the variation of the concentrations of water and nutrients, the larger the mean size of leaves the higher the resources consumed by the plant and the higher the decrease in concentration.

Appendix A.3. Plant Behaviour Example

In order to illustrate a prototypical plant behaviour we present in Figure A.6 an example of a plant evolution under a specific treatment. The example shows the evolution of the number and sizes of leaves and their respective color variations with incomes of water and nutrient. In particular, we present a situation in which the plant is first supplied with normal amounts of water and nutrients and then situate the plant under nutrient stress. Notice the variation in leaf generation, where

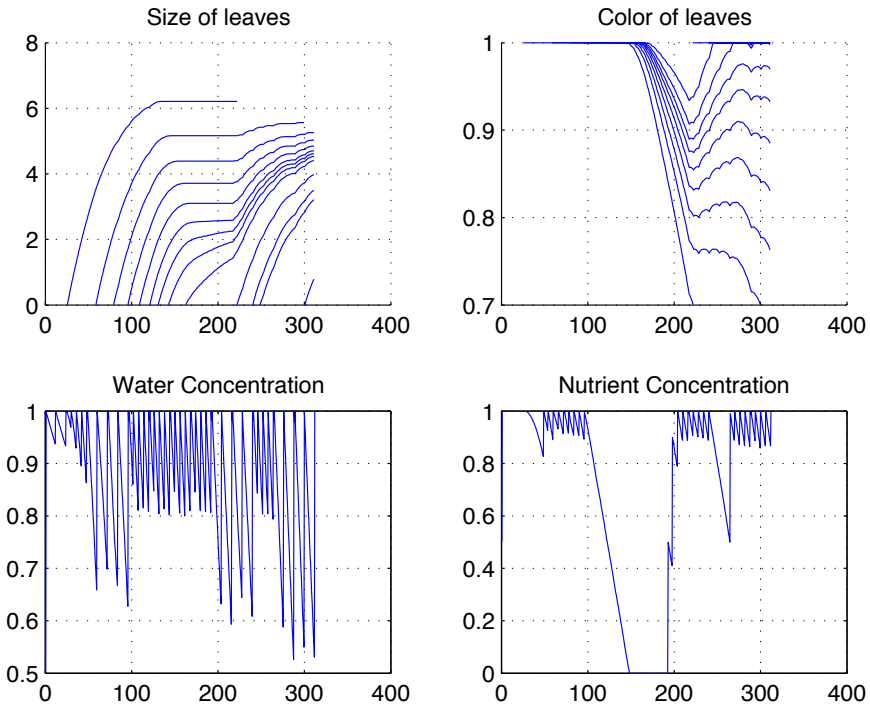


Figure A.6: Plant evolution example.

many leaves are generated at the beginning, when the plant is treated normally, to later diminish due to the nutrient stress. It is important to note that the plant simulator also emulates the permanent wilting point, where older yellow leaves cannot be fully recovered in the long run, even though they may show some temporal recovery, when their color gets below 0.9. This is so even when the nutrient supply is restored. These leaves are finally eliminated from the plant in the long run.

Appendix A.4. Plant Simulator Set-up

The delayed response of the plant is simulated using a delay of $\tau = 25$ hours. We set this relative high value for the delay in order to increase the weakness of the correlations between causes and effects. The threshold for the average saturation of water and nutrient in the generation of leaves is set to $thr_{gen} = 0.1$ and the threshold for the elimination of leaves according to their color is set to

$thr_{color} = 0.7$ (see Sec. Appendix A.1). For the simulation we use the Euler method with a differential for time of $dt = 0.5$ hours and an actuation interval of one hour. All the parameters of the equations of size and color variation rates were defined by exhaustive manual tuning (Table A.20) based on observations of Tobacco plants growth.

Agostini, A., Torras, C., Wörgötter, F., 2011. Integrating Task Planning and Interactive Learning for Robots to Work in Human Environments. In: Proceeding of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11). Barcelona, Spain. pp. 2386–2391.

Benson, S., 1995. Inductive learning of reactive action models. In: Proc. of the Twelfth Int. Conf. on Machine Learning. Morgan Kaufmann, pp. 47–54.

Finnie, G., Sun, Z., 2002. Similarity and metrics in case-based reasoning. International journal of intelligent systems 17 (3), 273–287.

Fourcaud, T., Zhang, X., Stokes, A., Lambers, H., Körner, C., 2008. Plant growth modelling and applications: the increasing importance of plant architecture in growth models. Annals of Botany 101 (8), 1053–1063.

Furnkranz, J., Flach, P., 2003. An analysis of rule evaluation metrics. In: Proc. of the Twentieth Int. Conf. on Machine Learning. Vol. 20. pp. 202–209.

GARNICS, 2010-2013. GARdeNIng with a Cognitive System. http://www.garnics.eu/.

Gil, Y., 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In: Proceedings of the Eleventh International Machine Learning Conference (ICML). Rutgers, NJ, pp. 87–95.

Table A.20: Plant Simulation Parameters

Leaf Size	Leaf Color	Stem Size	W. Sat.	N. Sat.
k_{n_l} = 1.4e-4	k_G = 1.9e-4	k_H = 2e-5	k_W =1e-4	k_N = 1.3e-4
k_S = 2.3e-5	k_{W_C} = 1.1e-4	k_{W_H} = 1.9e-4	k_V = 1e-4	$k_{\bar{S}_N}$ = 2.1e-4
k_{W_S} = 1.5e-4	k_{N_C} = 2.3e-4	k_{N_H} = 0.6e-4	$k_{\bar{S}_W}$ = 2.1e-4	
k_{N_S} = 0.8e-4	k_{L_C} = 1e-4			
k_{L_S} = 1.2e-4				
k_{T_S} = 3.4e-4				
k_{M_S} = 1.1e-4				

Hackett, C., 1973. An exploration of the carbon economy of the tobacco plant i. inferences from a simulation. Australian Journal of Biological Sciences 26 (5), 1057–1072.

Kang, M., Dumont, Y., Guo, Y., 2012. Proceedings pma12: The fourth international symposium on plant growth modeling, simulation, visualization and applications, shanghai, china, 31 october-3 november 2012.

LaValle, S., 2006. Planning algorithms. Cambridge Univ Pr.

McCallum, A., 1996. Reinforcement learning with selective perception and hidden state. Ph.D. thesis, The University of Rochester.

Mitchell, T. M., 1997. Machine learning. 1997. Burr Ridge, IL: McGraw Hill 45.

Paine, C., Marthews, T. R., Vogt, D. R., Purves, D., Rees, M., Hector, A., Turnbull, L. A., 2012. How to fit nonlinear plant growth models and calculate growth rates: an update for ecologists. Methods in Ecology and Evolution 3 (2), 245–256.

Petrick, R., Bacchus, F., 2002. A knowledge-based approach to planning with incomplete information and sensing. In: AIPS. pp. 212–221.

Shen, W., 1989. Rule creation and rule learning through environmental exploration. In: Proc. of the Eleventh Int. Joint Conf. on Artificial Intelligence. Morgan Kaufmann, pp. 675–680.

Verkroost, A., Wassen, M., 2005. A simple model for nitrogen-limited plant growth and nitrogen allocation. Annals of Botany 96 (5), 871–876.

Walsh, T., Littman, M., 2008. Efficient learning of action schemas and web-service descriptions. In: Proc. of the Twenty-Third AAAI Conf. on Artificial Intelligence. pp. 714–719.

Wang, X., 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In: ICML. pp. 549–557.

Wang, X., 1996. Planning while learning operators. In: AIPS. pp. 229–236.