

CONSTRUCTIVE INDUCTION OF GOAL CONCEPTS FROM AGENT PERCEPTS AND REINFORCEMENT FEEDBACK

Jurgita Kapočiūtė-Dzikienė, Gailius Raškinis

*Vytautas Magnus University
Vileikos St. 8, LT-3035 Kaunas, Lithuania
e-mail: j.kapociute@if.vdu.lt, g.raskinis@if.vdu.lt*

Abstract. This paper investigates the ability of an agent to recognize unseen goal states in an observable grid-world environment. This ability is important in order to allow the explicit planning of agent's action sequences. Grid-world states are described by the collection of attributes instead of "atomic" labels. This attribute-based state representation allows generalizing over the attributes and discriminating goal (reinforced) states from non-goal states. We argue that a biased induction technique is required to solve this supervised learning problem. A constructive induction technique is proposed which is able to discover goal concepts relevant to the grid-world environments, where training instances are 2D image-like objects. The set of constructive operators is defined and a search procedure in the space of sequences of these operators is described. The proposed technique was tested on two non-trivial sample tasks and was successful in learning goal concepts. It allowed synonymous descriptions of the same goal concept to be learnt depending on the set of operators and their activation order. The complete assessment of prospects and limits of the proposed constructive induction technique still needs to be done.

Keywords: reinforcement, supervised, learning, bias, constructive induction, operators, agent, grid-world, simulation, adaptive behavior.

1. Introduction

An adaptive agent is the system that perceives its environment and acts upon it continuously updating its internal configuration in order to improve its performance over time. The interaction of an agent and its environment is often simulated by the sequence of

iterations each consisting of the following steps: percept information reaches the agent through its sensors, the agent updates its internal configuration, the agent selects some action and performs it through its effectors. The action alters the state of the environment resulting in new sensory data. This is the beginning of the new percept-action iteration (Figure 1).

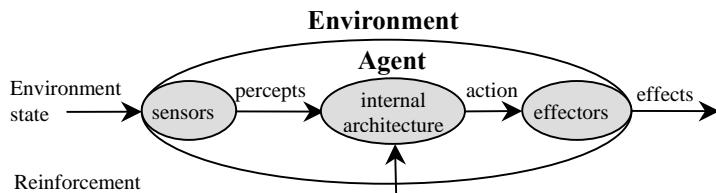


Figure 1. Illustration of an agent embedded into its environment

Reinforcement learning framework is the most widespread approach to simulate agent behavior. It assumes that there are some states¹ of the environment that are desirable to the agent. The information about the desirability of the current environmental state reaches the agent through the dedicated reinforcement channel (Figure. 1). The agent is expected to maxi-

mize its lifelong cumulative reinforcement, i.e. the agent is successful if it spends more and more time in these desirable states or if it finds them better.

Imagine one particular task in which the agent (A) perceives a grid-world environment containing some boxes (B) and trees (T). It disposes of a few actions such as going, jumping and pushing boxes in four directions. The agent is rewarded only if it collects all boxes into one cluster. An illustrative sequence of agent experiences in this world is shown in Figure 2.

¹ In this paper, we assume that the environment is observable, thus state = percepts

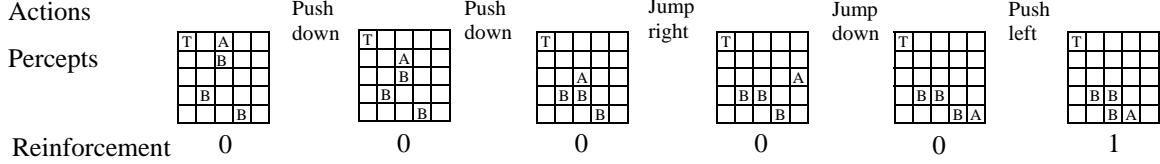


Figure 2. The sequence of experiences of an agent (A) in a grid-world environment containing trees (T) and boxes (B). The reinforcement is provided only if all boxes form one cluster

Traditional reinforcement learning techniques such as TD-learning [15], Q-learning[17] and ADP[12] model the above and similar tasks as a finite-state Markov decision process (MDP). They learn utility values of states or state-action pairs. Their action selection policy recommends an action that promises maximum utility in the forthcoming state. The explicit planning of action sequences consisting of more than one action is not much studied. The first reason for this is that many reinforcement learning techniques are world model-free, i.e. agents do not construct the knowledge body that would answer their questions of the type “what will happen if I perform this or that action?” The second reason is that the environment states are assigned “atomic” labels in contrast to the “distributed” state representation where states are described by some set of attributes. The attribute-based state representation is a necessary condition if we expect an agent to generalize over state attributes and discriminate unseen goal (reinforced) states from non-goal states.

The first limitation related to the construction of a world model has been addressed in our previous paper [4]. An adaptive agent called LEAD1, which had a world model as a part of its internal configuration, was proposed. In this paper, we address the second limitation, i.e. we investigate the feasibility of inducing abstract goal descriptions from the sets of reinforced and unreinforced states described by their attributes.

More formally the problem can be stated as follows. Given two collections P and R , where P is the collection of percepts $P = \langle p_1, p_2, \dots, p_n \rangle$, $p_k = \{p_{ij}\}_k$ and $R = \langle r_1, r_2, \dots, r_n \rangle$, $r_k \in \{0,1\}$ is the collection of reinforcements corresponding to these percepts, find the binary function F , such that $F(p_k) = r_k$, for every p_k . This is the standard description of a supervised learning problem. It may seem that any supervised learner, e.g. k-DNF learner, may suffice to solve it. In fact, k-DNF learner may be sufficient to solve problems where reinforcement value (class) is the function of a few particular attributes, i.e. it depends on the contents of particular cells in a grid-world environment. Such a case is illustrated by Figure 3.

However, the majority of goals that are worth pursuing in a grid world environment are too hard for any k-DNF learner. In case of the goal “boxes form one cluster” (Figure. 2), there exists a huge number of possibilities of arranging boxes in a grid-world so they form one cluster. Any decision tree or a rule set produced by the k-DNF learner would represent a

memorization of past reinforced training instances rather than an abstract concept. Thus, we propose a solution based on constructive induction. The goal concept F would be sought as a composite function of pre-defined operators suitable for processing of 2D image-like grid-world percepts.

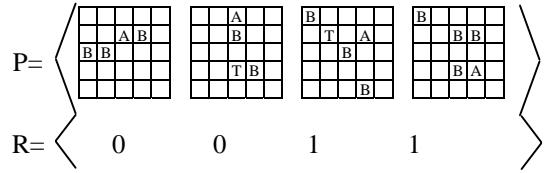


Figure 3. Illustration of an easy task for a k-DNF learner. The goal concept is $p_{11} = B$

2. Related work in image processing and constructive induction

The research in the field of image classification focuses on the discrimination of pixel-based images. These images often represent noisy real-world objects such as printed or handwritten characters, faces, facial expressions, etc. Many efficient problem-specific filters were developed for noise reduction, image enhancement, feature detection, image segmentation and other low-level image processing tasks [11]. Once a researcher defines a problem-specific sequence of low-level image processing steps and compiles a training set of pre-classified images, neural networks or statistical machine learning techniques are used to solve the image classification task. Grid world images are not noisy. They are also truly discrete in nature in comparison to semi-continuous pixel-based graphics. However, the most important reason of why image classification approach seems to be inadequate to solve our goal concept induction problems is the desired autonomy of an agent. The low-level image processing, in fact, can be seen as the transformation of image representation space. This transformation is defined by a human researcher while we expect it could be discovered by an agent itself.

Constructive induction is a double-search [1] process. The search is performed both to find the transformation of the object representation space and to discover generalizations in this new transformed space. The transformation usually involves the construction of new attributes by combining existing attributes by arithmetical, logical or domain-specific operators. The early constructive induction methods are surveyed by Wnek et al. [19] and Callan [2]. As application-specific

operator sets make constructive induction methods application-specific, none of the existing approaches is suitable to solve our problem. For instance, BACON [6] and STAGGER [13] were not designed for processing image-like objects while the method proposed by Maksimov [7] was designed to process pixel-based graphic images. Our technique is mostly inspired by the latter method and borrows the idea of recombination, i.e. constructing new attributes as functions of old attributes from FRINGE [10] and CITRE [8], the idea of introducing Boolean attributes by clustering existing numeric attributes from STAGGER [13], and the idea of object pattern transformations, i.e. changing one pattern into another from STABB [16].

The latest researches are orientated on constructive induction application in different areas, besides it classification relies on genetic optimization and search techniques: Otero et al.[9], Krawiec[5], Shafti et al.[14], Weninger et al. [18], etc. (for review see [3]). Image processing programs consisting of operator sequences are encoded as gene strings. Natural selection mechanisms are applied to the population of these gene strings in order to find good image processing programs. The above methods focus on the optimization of genetic search strategy, while our technique would be purely symbolic and rely on exhaustive search.

3. Algorithm description

The input to the proposed algorithm consists of two collections P and R , where P is the collection of percepts $P = \langle p_1, p_2, \dots, p_n \rangle$, $p_k = \{p_{ij}\}_k$ and $R = \langle r_1, r_2, \dots, r_n \rangle$, $r_k \in \{0,1\}$ is the collection of reinforcements corresponding to these percepts. The objective of the algorithm is to find a binary function F such that $F(p_k) = r_k$ for all p_k . The search for such a function is performed in a breadth first-manner in the space of possible operator sequences. The search space is defined both by the operator set and by their compatibility constraints.

3.1. Constructive operators

Constructive operators manipulate objects of four basic types: Boolean value (L), integer value (N), Boolean matrix (M) and percept matrix (I). Operators accept one or two objects as their input and produce a single object as their output. Some operators may decompose Boolean matrix type objects into a few components of the same type. This set of components still represents a single though compound object. This compound type is denoted M_m , with the subscript m indicating the number of components in the object.

All objects are organized in super-structures called collections. The collection of objects of some basic type T is denoted as $\langle T \rangle$. Operators are subdivided into two groups: intermediate operators and terminal operators. Intermediate operators are applied to the collection on an object-by-object basis. Terminal operators

are applied to the whole collection at once. The list of constructive operators is given in Table 1.

3.2. Search strategy

Object collections are stored in the queue. The top collection of the queue is the collection P of positively and negatively reinforced percepts. All subsequent collections are derived in a breadth-first search manner from this top collection by applying constructive operators. Search is terminated when some collection of the queue matches the collection of reinforcements R .

An operator can be applied to some collection (or a pair of collections in the case of a 2-argument operator) in the queue only if all objects in this collection(s) match operator's input type declaration. Intermediate operators are applied to the collection on an object-by-object basis. Terminal operators are applied to the whole collection at once. The derived collection always has the same number of objects as its parent collection(s) and thus the same number of objects as the initial top collection.

Collections are assigned a depth value which represents the number of transformation steps (operators) that were necessary to produce that collection. The depth of the top collection $\text{depth}(P)$ is 0. The depth of some derived collection $col_{out} \leftarrow \text{Op}(col_{in1}, col_{in2})$ is given by:

$$\text{depth}(col_{out}) = 1 + \max(\text{depth}(col_{in1}), \text{depth}(col_{in2})).$$

The search is terminated if there isn't any solution found within collections bounded by max_depth .

The outline of our constructive induction technique INDUCE is given by the pseudo code below. An illustration of the same process is given by Figure 4.

4. Investigation of illustrated samples

Our algorithm of constructive induction was investigated on a pair of basic problems. These investigations had an objective to test the feasibility of constructive induction approach for discovering goal concepts. The complete evaluation of all possibilities and limits of the proposed induction technique would require further testing.

An agent was placed in a fully observable grid-world environment. Grid-world cells could take values from the set $\{A, B, T, \emptyset\}$ denoting an agent, a box, a tree, and an empty cell respectively. Agent was making random moves, moves recommended by its teacher or by its planner component [4]. Percepts and their respective reinforcement values were stored until positive (1) and negative (0) reinforcement was first received and supervised learning was possible. Then constructive induction algorithm was called for the first time to induce the goal concept description F . Since then, the goal concept description was continuously tested against new incoming percept-reinforcement input pairs. If the concept description F incorrectly predicted some reinforcement value, it was discarded and reconstructed in a batch-learning manner on the basis of extended experience.

The agent was asked to learn two goal concepts “the agent is next to some tree” (F1) and “the largest cluster of boxes is next to some tree” (F2). The agent succeeded in learning both concepts (Figures 5, 6).

Table 1. Constructive operators

Operator	Illustration
Intermediate operators	
$M \leftarrow \text{Change}\{El\}(I)$, $El \in \text{Nom}^2$	$I = \begin{array}{ c c c } \hline C & & \\ \hline B & B & A \\ \hline B & & B \\ \hline C & & \\ \hline \end{array}$ $\xrightarrow{\text{Change}\{B\}}$ $M = \begin{array}{ c c c } \hline \text{■} & \text{■} & \\ \hline \text{■} & \text{■} & \text{■} \\ \hline \text{■} & \text{■} & \text{■} \\ \hline \text{■} & \text{■} & \\ \hline \end{array}$ <p>Replaces the value of El by 1 and other values by 0. Grey squares denote <i>true</i> (1) and white squares denote <i>false</i> (0) value.</p>
$L \leftarrow \text{Conjunction}(L, L)$ $L \leftarrow \text{Disjunction}(L, L)$ $L \leftarrow \text{Negation}(L)$ $L \leftarrow \text{Equivalence}(L, L)$	Return the result of logical conjunction (\wedge), disjunction (\vee), negation (\neg), and equivalence (\Leftrightarrow).
$L \leftarrow \text{Compare}(M_m, M_m)$	$M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $\xrightarrow{\text{Compare}}$ $L = 1$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $\xrightarrow{\text{Compare}}$ $L = 1$ <p>Returns 1 if all the respective components of both input arguments are equal.</p>
$M_m \leftarrow \text{ObjConj}(M_m, M_m)$	$M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $\xrightarrow{\text{ObjConj}}$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $\xrightarrow{\text{ObjConj}}$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ <p>Returns an object consisting of component-by-component conjunction of both input arguments.</p>
$M_m \leftarrow \text{ObjDisj}(M_m, M_m)$	$M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $\xrightarrow{\text{ObjDisj}}$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $\xrightarrow{\text{ObjDisj}}$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ <p>Returns an object consisting of component-by-component disjunction of both input arguments.</p>
$M_m \leftarrow \text{MultipleConj}(M, M_m)$	$M = \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ $\xrightarrow{\text{MultipleConj}}$ $M_2 = \left\{ \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array}, \begin{array}{ c c } \hline \text{■} & \text{■} \\ \hline \text{■} & \text{■} \\ \hline \end{array} \right\}$ <p>Returns an object consisting of components obtained via conjunction of the first input argument with all the components of the second argument.</p>

² Nom is the set of possible values appearing in the grid-world cells

$M_m \leftarrow \text{MultipleDisj}(M, M_m)$	<p>MultipleDisj</p> <p>$M = \begin{array}{ c c c c } \hline & & & \\ \hline & & \text{True} & \\ \hline & & & \\ \hline \end{array}$</p> <p>$M_m = \begin{array}{ c c } \hline & \text{True} \\ \hline \end{array}$</p> <p>$M_2 = \begin{array}{ c c c c } \hline & & & \\ \hline & & \text{True} & \\ \hline & & & \\ \hline \end{array}$</p>	Returns an object consisting of components obtained via disjunction of the first input argument with all the components of the second argument.
$M \leftarrow \text{SelectiveDisj}(L_m, M_m)$	<p>SelectiveDisj</p> <p>$L_3 = \{1, 0, 1\}$</p> <p>$M_3 = \begin{array}{ c c c } \hline & \text{True} & \\ \hline \text{True} & & \\ \hline & & \text{True} \\ \hline \end{array}$</p> <p>$M = \begin{array}{ c c c } \hline & \text{True} & \\ \hline \text{True} & & \\ \hline & & \text{True} \\ \hline \end{array}$</p>	Returns an object consisting of disjunction of selected components of the second argument.
$N \leftarrow \text{CountTVal}(M)$	<p>CountTVal</p> <p>$M = \begin{array}{ c c c } \hline & \text{True} & \\ \hline \text{True} & & \\ \hline & & \text{True} \\ \hline \end{array}$</p> <p>$N = 3$</p>	Counts the number of <i>true</i> values in the input argument.
$N \leftarrow \text{CountComponents}(M_m)$	<p>CountComponents</p> <p>$M_3 = \begin{array}{ c c c } \hline & \text{True} & \\ \hline \text{True} & & \\ \hline & & \text{True} \\ \hline \end{array}$</p> <p>$N = 3$</p>	Counts the number of components in the input argument.
$L \leftarrow \text{TValExists}(M)$	<p>TValExists</p> <p>$M = \begin{array}{ c c c } \hline & \text{True} & \\ \hline \text{True} & & \\ \hline & & \text{True} \\ \hline \end{array}$</p> <p>$L = 1$</p>	Tests if at least one <i>true</i> value is found in the input argument.
$L_m \leftarrow \text{MaxComponent}(M_m)$	<p>MaxComponent</p> <p>$M_4 = \begin{array}{ c c c c } \hline & \text{True} & & \\ \hline \text{True} & & & \\ \hline & & \text{True} & \\ \hline & & & \text{True} \\ \hline \end{array}$</p> <p>$L_4 = \{1, 0, 0, 0\}$</p>	Finds the index of the first component having maximum number of <i>true</i> values.
$L_m \leftarrow \text{MaxComponents}(M_m)$	<p>MaxComponents</p> <p>$M_4 = \begin{array}{ c c c c } \hline & \text{True} & & \\ \hline \text{True} & & & \\ \hline & & \text{True} & \\ \hline & & & \text{True} \\ \hline \end{array}$</p> <p>$L_4 = \{1, 0, 1, 0\}$</p>	Finds indices of all components having maximum number of <i>true</i> values.
$L_m \leftarrow \text{MinComponent}(M_m)$	<p>MinComponent</p> <p>$M_4 = \begin{array}{ c c c c } \hline & \text{True} & & \\ \hline \text{True} & & & \\ \hline & & \text{True} & \\ \hline & & & \text{True} \\ \hline \end{array}$</p> <p>$L_4 = \{0, 1, 0, 0\}$</p>	Finds the index of the first component having minimum number of <i>true</i> values.
$L_m \leftarrow \text{MinComponents}(M_m)$	<p>MinComponents</p> <p>$M_4 = \begin{array}{ c c c c } \hline & \text{True} & & \\ \hline \text{True} & & & \\ \hline & & \text{True} & \\ \hline & & & \text{True} \\ \hline \end{array}$</p> <p>$L_4 = \{0, 1, 0, 1\}$</p>	Finds indices of all components having minimum number of <i>true</i> values.
$M_m \leftarrow \text{Put4Neighbors}(M_m)$	<p>Put4Neighbours</p> <p>$M_2 = \begin{array}{ c c } \hline \text{True} & \\ \hline & \\ \hline \end{array}$</p> <p>$M_2 = \begin{array}{ c c c c } \hline & \text{True} & & \\ \hline \text{True} & & & \\ \hline & & \text{True} & \\ \hline & & & \text{True} \\ \hline \end{array}$</p>	Extends <i>true</i> values to their neighborhood: if $p_{ij}=1$ then $p_{i\pm 1,j}=1$ and $p_{ij\pm 1}=1$.
$M_m \leftarrow \text{Put8Neighbors}(M_m)$	<p>Put8Neighbours</p> <p>$M_2 = \begin{array}{ c c } \hline \text{True} & \\ \hline & \\ \hline \end{array}$</p> <p>$M_2 = \begin{array}{ c c c c } \hline & \text{True} & & \\ \hline \text{True} & & & \\ \hline & & \text{True} & \\ \hline & & & \text{True} \\ \hline \end{array}$</p>	Extends <i>true</i> values to their neighborhood: if $p_{ij}=1$ then $p_{i\pm 1,j}=1$, $p_{ij\pm 1}=1$, $p_{i\pm 1,j\pm 1}=1$.

$M_m \leftarrow \text{SplitToGroups4}(M)$	$M = \begin{array}{ c c c c } \hline & & & \\ \hline & & \text{█} & \\ \hline & \text{█} & & \\ \hline & & & \\ \hline \end{array} \xrightarrow{\text{SplitToGroups4}} M_2 = \left\{ \begin{array}{ c c } \hline & \text{█} \\ \hline & \\ \hline \end{array}, \begin{array}{ c c c c } \hline & & & \\ \hline \end{array} \right\}$	Splits an object into spatially disconnected components. The connectedness is defined as in Put4Neighbors operator.
$M_m \leftarrow \text{SplitToGroups8}(M)$	$M = \begin{array}{ c c c c } \hline & & & \\ \hline & & \text{█} & \\ \hline & \text{█} & & \\ \hline & & & \\ \hline \end{array} \xrightarrow{\text{SplitToGroups8}} M_1 = \begin{array}{ c c } \hline & \text{█} \\ \hline & \\ \hline \end{array}$	Splits an object into spatially disconnected components. The connectedness is defined as in Put8Neighbors operator.
$M_m \leftarrow \text{SplitToGroups}(M)$	$M = \begin{array}{ c c c c } \hline & & & \\ \hline & & \text{█} & \\ \hline & \text{█} & & \\ \hline & & & \\ \hline \end{array} \xrightarrow{\text{SplitToGroups}} M_3 = \left\{ \begin{array}{ c c } \hline & \text{█} \\ \hline & \\ \hline \end{array}, \begin{array}{ c c } \hline & \\ \hline & \text{█} \\ \hline & \\ \hline \end{array}, \begin{array}{ c c } \hline & \\ \hline & \\ \hline & \text{█} \\ \hline & \\ \hline \end{array} \right\}$	Splits an object into components, where every <i>true</i> value is a separate component.
Terminal operators:		
$\langle L \rangle \leftarrow \text{Cluster}(\langle N \rangle)$	$\langle 0, 1, 4, 3, 4, 2, 1 \rangle \xrightarrow{\text{Cluster}} \langle 0, 0, 1, 1, 1, 0, 0 \rangle$	Clusters all integer values of the input collection into two clusters.
$\langle L \rangle \leftarrow \text{MaxObjects}(\langle N \rangle)$	$\langle 0, 1, 4, 3, 4, 2, 1 \rangle \xrightarrow{\text{MaxObjects}} \langle 0, 0, 1, 0, 1, 0, 0 \rangle$	Finds the indices of the greatest integer values in the input collection.
$\langle L \rangle \leftarrow \text{MinObjects}(\langle N \rangle)$	$\langle 0, 1, 4, 3, 4, 2, 1 \rangle \xrightarrow{\text{MinObjects}} \langle 1, 0, 0, 0, 0, 0, 0 \rangle$	Finds the indices of the smallest integer values in the input collection.

```

INDUCE(P, R, max_depth)
  // P – collection of percepts
  // R – collection of corresponding reinforcements
  // max_depth – maximum allowed length of operator sequence
  // Q – queue, initially empty
  Assign depth(P) := 0
  Add P to queue Q
  For depth d := 0 to max_depth
    For every collection coli ∈ Q such that depth(coli) = d
      For every intermediate 1-argument operator Op(type)
        IF all objects ok ∈ coli match the type THEN
          Append new collection colnew to the end of the queue Q
          Fill colnew with objects Op(ok) for all ok ∈ coli
          Assign depth(colnew) := 1 + d
          Apply TERMINAL_OPERATORS(colnew, R)
        For every collection colj ∈ Q such that depth(colj) ≤ d
          For every intermediate 2-argument operator Op(type1, type2)
            IF all pairs of objects {ok, ol} ok ∈ coli, ol ∈ colj
              match {type1, type2} THEN
                Append new collection colnew to the end of the queue Q
                Fill colnew with objects Op(ok, ol) for all ok ∈ coli, ol ∈ colj
                Assign depth(colnew) := 1 + d
                Apply TERMINAL_OPERATORS(colnew, R)
      END
    TERMINAL_OPERATORS(col, R)
      IF col=R THEN TERMINATE(success)
      For every terminal operator Op
        IF col is of the type <N> THEN
          Create new collection colnew := Op(col)
          Append colnew to the end of the queue Q
          Assign depth(colnew) := 1 + depth(col)
        IF colnew=R THEN TERMINATE(success)
  END

```

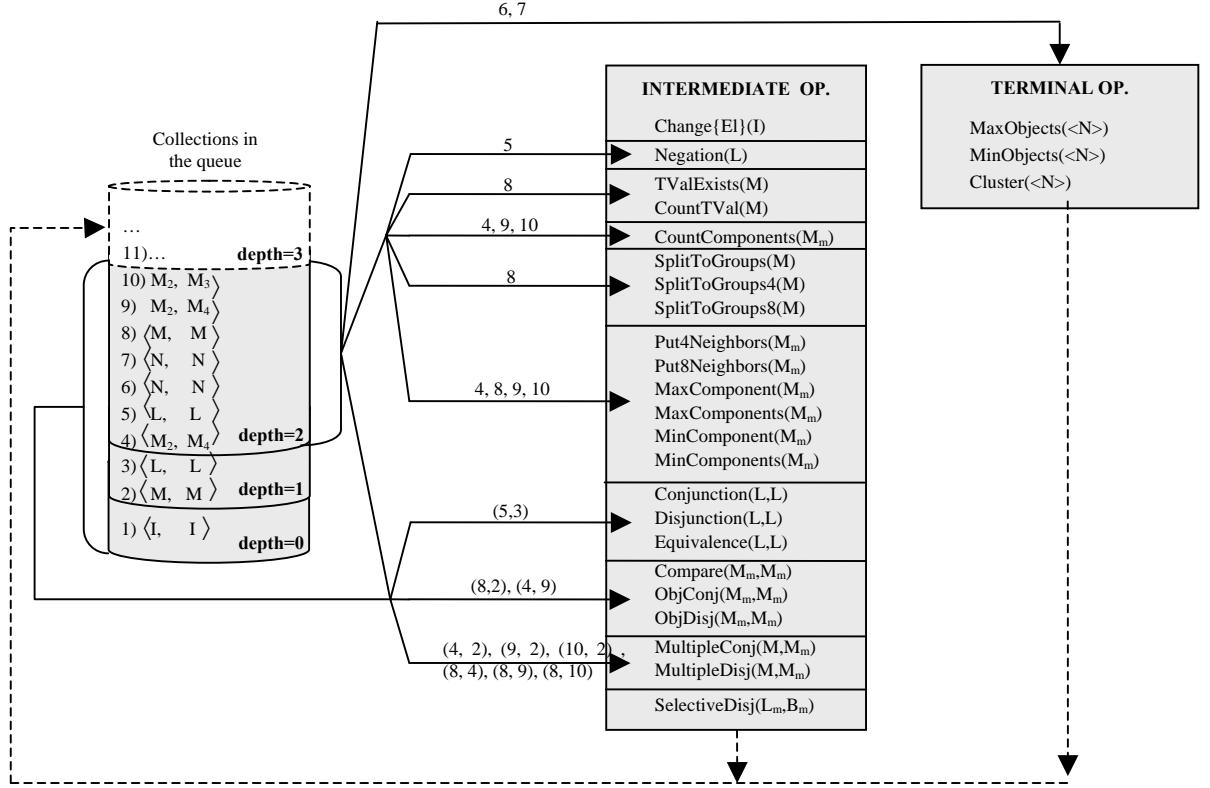


Figure 4. Illustration of the search process at depth $d = 2$. Solid arrows lead to operators that are activated because of the collections matching their input types are found in the queue. Operator arguments (collection indices) are indicated beside the arrows. The order of operator activation is top-to-bottom

$$F_I(P) \equiv TValExists(ObjCon(Put4Neighbors(Change\{A\}(P)), Change\{T\}(P)))$$

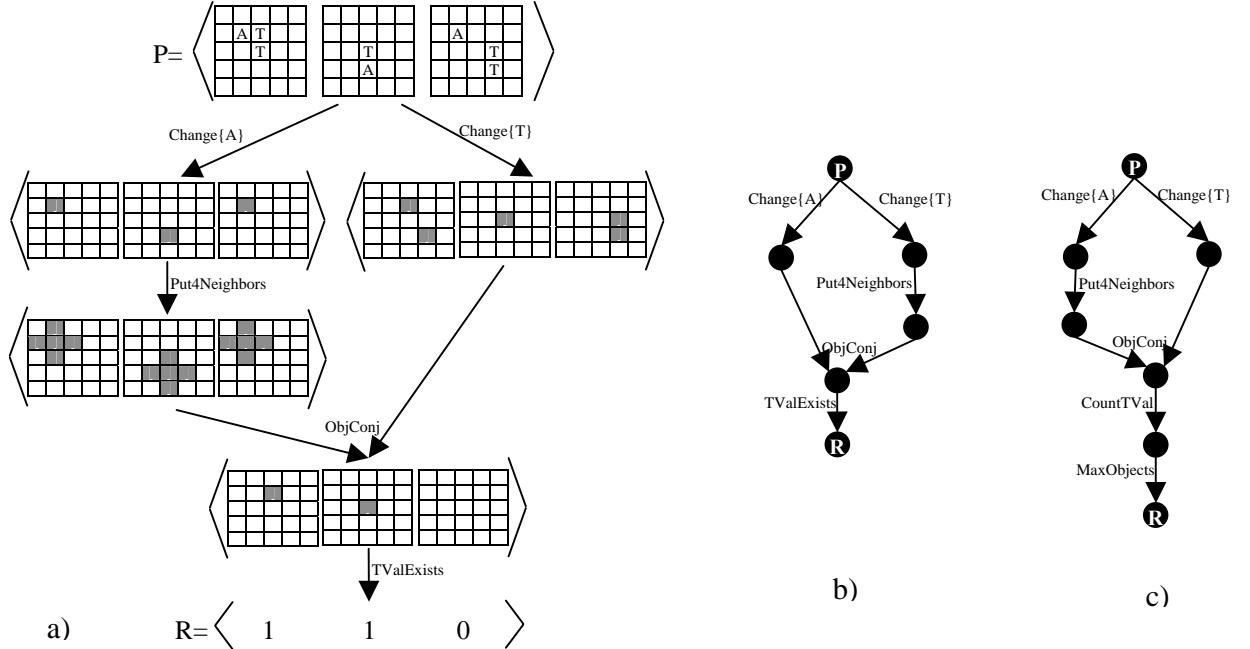


Figure 5. Learning the concept “the agent is next to some tree”. The graphs represent induced goal concepts. Nodes of the graphs correspond to object collections in the queue. Arrows represent the transformations of object collections by constructive operators. a)The solution graph F_I displaying the temporary content of object collections. b) A slightly different solution obtained by changing operator ordering, i.e. moving operator $Change\{A\}$ before $Change\{T\}$. c) One more solution obtained after the operator $TValExists$ has been eliminated from the list of possible operators

$$F_2(P) \equiv TValExists(ObjConj(SelectiveDisj(MaxComponent(SplitToGroups4(Change\{B\}(P))), \\ SplitToGroups4(Change\{B\}(P))), Put4Neighbors(Change\{T\}(P))))$$

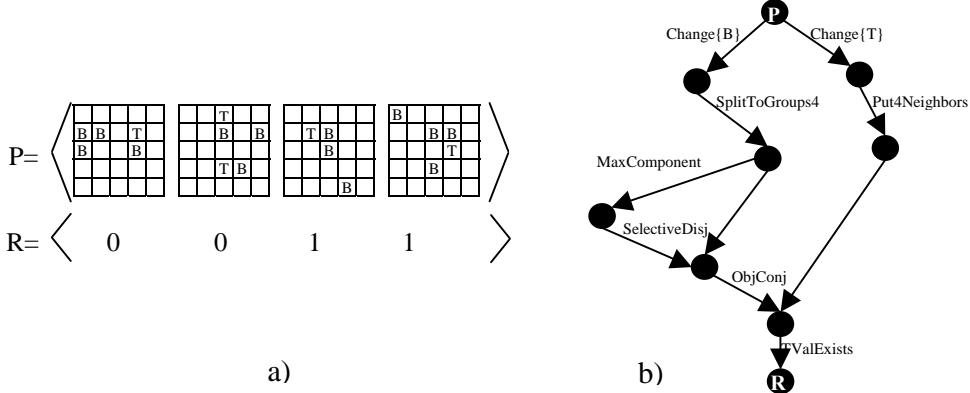


Figure 6. Learning the concept “the largest cluster of boxes is next to some tree”. a) The excerpt from the training collections P and R . b) The induced goal concept F_2 . Nodes of the graph correspond to object collections in the queue. Arrows represent the transformations of object collections by constructive operators

The investigations revealed that:

- The proposed induction technique allows synonymous ways of learning the same goal concept. Which of these synonymous descriptions would be found depends on the list of operators and their activation order.
- The extent of the set of concepts learnable by the proposed induction technique depends on the list of operators and the search depth. For instance, the lack of the operator *SplitToGroups4* would have inhibited learning the second goal concept. If the value of *max_depth* had been limited to 3, none of the concepts would have been learnt.

It is interesting to mention that the task presented by fig. 3, which is very easy for any k-DNF learner, is not solvable by the present technique unless operator “extracting” grid cell values at particular positions (pixels) would be added to the operator set.

5. Discussion and conclusions

The constructive induction technique, proposed in this paper, is search-based like many other symbolic induction techniques. However, it is biased for discovering goal concepts relevant to grid-world environments where training instances are 2D image-like objects, i.e. where the attributes of the training instances relate to each other as elements of a 2D spatial matrix. This paper demonstrated that the proposed technique can learn a few non-trivial goal concepts. It allowed synonymous ways of learning the same goal concept depending on the list of operators and their order.

The described technique was integrated into the agent called LEAD1 [4]. The planner of LEAD1 sought for the shortest action sequence leading to some reinforced environment state. The planner

followed a forward breadth-first strategy on the basis of the agent’s current world model. Expected outcomes of every action sequence (states) were submitted to the goal description (constructed off-line by the present constructive induction technique), which served as a terminating condition for the action planner.

While the agent has no knowledge of the consequences of its actions and no knowledge of the goal concept, it might be quite difficult to achieve the first goal by just trying random action sequences. For instance, the goal “boxes form one cluster” (Figure. 2) would be hard to achieve by random actions in a sufficiently large environment. Without positively reinforced states the constructive induction technique has nothing to learn, and any planner wanting for the goal concept is “blind” as a result. Thus, an agent would need some guidance from the teacher in selecting its actions for achieving non-trivial goals. We assume this is not an important limitation to our approach because it is paralleled in living biological agents.

Training set for our constructive induction technique (when it is embedded into an agent) consists overwhelmingly of non-reinforced states. It would be interesting to investigate the strategies about which non-reinforced training samples should be kept in agent’s memory and which ones should be dropped from it. Another interesting and important research question is the question of the relationship of the operator set to the extent of the tasks that can be solved by our constructive induction technique as well as related questions of minimum operator set and of completeness of the operator set for grid-world environments.

References

- [1] **E. Bloedorn, R.S. Michalski.** Data-Driven Constructive Induction in AQ17-PRE: A Method and Experiments. *Proceedings of the Third International Conference on Tools for AI*, 1991, 30-37.
- [2] **J. Callan.** Use of Domain Knowledge in Constructive Induction. *Technical report UM-CS-1990-095. University of Massachusetts, Amherst, MA, USA*, 1991.
- [3] **C. Estebanez, R. Aler.** Generating Automatic Projections by Means of Genetic Programming. *Alba, E., Blum, C, Isasi, P. Optimization Techniques for Solving Complex Problems (Eds.)*, 2009, 3-14.
- [4] **J. Kapočiutė-Dzikienė, G. Raškinis.** Learning Explicit Action Definitions in Deterministic Observable Grid-World Environment. *Informatica, MII, Vilnius, Lithuania*, 2009 (*in review*).
- [5] **K. Krawiec.** Constructive Induction in Learning of Image Representation. *Institute of Computing Science Poznan University of Technology Research Report RA-006/2000, Institute of Computing Science, Poznan, Poland*, 2000.
- [6] **P. Langley, G. Bradshaw, H. Simon.** Rediscovering Chemistry with the BACON system. *Machine learning: An artificial intelligence approach (chapter 10)*. Palo Alto, CA, USA, 1983.
- [7] **V.V. Maksimov.** System, learning to classify geometrical pictures. Smirnov, M.S. Modelirovanije. *Obucenija i povedenija (Eds.)*, Akademija nauk, 1975, 29-151 (*in Russian*).
- [8] **C.J. Matheus, L.A. Rendell.** Constructive induction on decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, 645-650.
- [9] **F.E.B. Otero, M.M.S. Silva, A.A. Freitas, J.C. Nievola.** () Genetic programming for attribute construction in data mining. In C. Ryan et al., (Eds), *Genetic Programming (Proceedings of EuroGP'03)*, Vol. 2610, 2003, 389–398.
- [10] **G. Pagallo.** Learning DNF by decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, 639-644.
- [11] **R. Poli.** Genetic Programming for Image Analysis, *Technical Report CSRP-96-1, The University of Birmingham, UK*, 1996.
- [12] **S.J. Russell, P. Norvig.** Artificial Intelligence: A Modern Approach (2nd edition). Pearson Education, Publishers Inc. Upper Saddle River, New Jersey, USA, 2003.
- [13] **J.C. Schlimmer.** Incremental adjustment of representations. *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, 1987, 79-90.
- [14] **L.S. Shafit, E. Perez.** Constructive induction and genetic algorithms for learning concepts with complex interaction. In H.-G. Beyer et al., eds., *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'05)*, Washington, DC. ACM Press, New York, 2005, 1811–1818.
- [15] **R.S. Sutton.** Learning to predict by the methods of temporal differences. *Machine Learning*, Vol. 3, 1988, 9-44.
- [16] **P. E. Utgoff.** Shift of bias for inductive concept learning. *Machine learning: An artificial intelligence approach*, 1986, 107-147.
- [17] **C. Watkins, P. Dayan.** Q-learning. *Machine learning*, Vol. 8 (3), 1992, 279-292.
- [18] **T. Weninger, W.H. Hsu, J. Xia, W. Aljandal.** An Evolutionary Approach to Constructive Induction for Link Discovery. *Proceedings of the Eleventh Annual Conference Companion on Genetic and Evolutionary Computation Conference*, Montreal, Canada, 2009, 2167-2172.
- [19] **J. Wnek, R.S. Michalski.** Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning*, Vol. 14(2), 1994, 139-168.

Received January 2010.