Robotics and Autonomous Systems 83 (2016) 1–14

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Automatic discovery of relational concepts by an incremental graph-based representation

Ana C. Tenorio-González *, Eduardo F. Morales

[1] Instituto Nacional de Astrofísica, Óptica y Electrónica, Luis Enrique Erro 1, Sta. Ma. Tonantzintla, Puebla, 72840, Mexico

HIGHLIGHTS

- We designed a method to learn relational concepts from a graph-based representation.
- Our method is designed to discover common/useful concepts of an environment.
- Our method can be used by an autonomous agent like a robot.
- Our method learned common concepts in three domains (polygons/furniture/floors).
- Independent human users validated the common concepts learned by our method.

ARTICLE INFO

Article history:
Received 8 December 2014
Received in revised form
11 May 2016
Accepted 27 June 2016
Available online 5 July 2016

Keywords:
Robot learning
Automatic concept learning
Concept discovery
Predicate invention
Inductive logic programming

ABSTRACT

Automatic discovery of concepts has been an elusive area in machine learning. In this paper, we describe a system, called ADC, that automatically discovers concepts in a robotics domain, performing predicate invention. Unlike traditional approaches of concept discovery, our approach automatically finds and collects instances of potential relational concepts. An agent, using ADC, creates an incremental graph-based representation with the information it gathers while exploring its environment, from which common sub-graphs are identified. The subgraphs discovered are instances of potential relational concepts which are induced with Inductive Logic Programming and predicate invention. Several concepts can be induced concurrently and the learned concepts can form arbitrarily hierarchies. The approach was tested for learning concepts of polygons, furniture, and floors of buildings with a simulated robot and compared with concepts suggested by users.

1. Introduction

The increasing capabilities with which robots are provided at present, are allowing them to perform different tasks in a better way. However, before robots can be used to perform simple tasks, it is normally necessary to provide them with data and complex programs designed by users to simplify the reasoning process of the robot. This can be a time consuming process and involves several iterations until the robot is able to achieve the intended goals. An alternative approach is to let the robot discover by itself the required concepts to accomplish its tasks. Robots can obtain useful data, directly from their own experience with the environment, that could be used to induce concepts. Such abstract concepts can be used to simplify robotics tasks, such as navigation, planning or reasoning. In this paper, we describe how concepts can be automatically learned by a robot while it is exploring its environment. Traditionally, concept discovery departs from a given set of examples. In this research, the agent automatically searches and collects instances of potential concepts.

Robot learning has been a very active research area. Most of the research has been based on reinforcement learning and programming by demonstration [1,2] and, to a smaller extend, on concept learning [3]. In reinforcement learning and programming by demonstration the robot learns how to perform a task and normally assumes that the state-space representation is specified beforehand by the user. Other researches have tried to learn concepts that can be used to represent states useful to a robot, however, the agent normally learns a single concept at a time and the user is heavily involved in carefully preparing the learning settings.

Among the most commonly used approaches for concept discovery are those based on Inductive Logic Programming (ILP) with

* Corresponding author.
E-mail addresses: catanace17@inaoep.mx (A.C. Tenorio-González), emorales@inaoep.mx (E.F. Morales).
[1] http://www.inaoe.mx.

http://dx.doi.org/10.1016/j.robot.2016.06.012
0921-8890/

predicate invention. ILP with predicate invention provides an understandable representation (to humans) and mechanisms to introduce new predicates not available in the initial background knowledge. It is based on a logic programming language with powerful inference mechanisms that cannot be found in other machine learning techniques. ILP systems using predicate invention can be classified into approaches based on reformulation and on demand-driven approaches [4]. Systems using reformulation introduce new predicates produced by combining or restructuring other predicates to produce a more compact and precise theory (e.g., [5–8]). The demand-driven approaches introduce new predicates when the vocabulary is not enough to induce a theory (e.g., [9–18]).

In this paper, we describe a new approach ADC (Automatic Discovery of Concepts) for discovery of concepts with demand-driven predicate invention in a robotic domain. The robot gathers information while exploring its environment, identifies similar instances of potential concepts and learns relational concepts using ILP. Unlike previous systems, the proposed algorithm is designed to learn multiple concepts about objects and their relations, building hierarchies of concepts, based on data incrementally obtained from the direct experience of an agent with an unknown environment. We applied the proposed approach to learn concepts involving polygons, furniture, and floors of buildings that could be used for manipulation and navigation tasks. We evaluated the usefulness of the induced concepts in new environments and compared them to the expected concepts of human users. The rest of the paper is organized as follows. Section 2 describes the close related work compared with ADC. In Section 3, the proposed system is described in detail. Section 4 presents the experimental results and Section 5 gives general conclusions and future research directions.

2. Related work

In most of ILP systems there is a strong dependency on the user, the data is in many cases, collected by the user before the learning process, the learning scenarios must be prepared, and usually, there is no simultaneous concept learning, i.e., only one concept is learned at a time. In this paper, we propose ADC algorithm to learn several concepts (also hierarchical concepts) at the same time with demand-driven predicate invention; what the system learns depends on the information it gathers during the exploration of an environment and it is not known in advance.

In particular, among the ILP approaches with demand-driven predicate invention, *Statistical Predicate Invention* (SPI) [19] addresses the discovery of new concepts and proposes a generalization of predicate invention, known as statistical learning for hidden variable discovery. The algorithm, *Multiple Relational Clustering* (MRC), is presented to cluster objects, attributes and their relations using Markov logic (an extension of FOL). The process of clustering is done automatically, and it is equivalent to the introduction of new predicates (predicate invention), where each cluster represents a unit predicate. In [16] an algorithm based on teleo-reactive programs (TRP) is proposed for learning new concepts and skills of different hierarchies from existing knowledge. TRP represents hierarchical procedural knowledge, based on reactive execution of goal-oriented skills. In the manner of a STRIPS planner, each skill consists of a goal, an initial state or precondition, an action or method to reach the goal, and a final state or postcondition. First a bottom-up inference mechanism is performed to identify the current state using the perceptions of the agent and the background knowledge, then TRP searches for the first high-level goal that is not satisfied, and tries to form a path in the hierarchy of skills from the current state of the agent to that goal. When a sequence of skills to achieve the goal cannot be found, the algorithm introduces new skills. The new skills form their preconditions and goals using preconditions of concepts and skills closest to the goal. SPI and TRP were tested on databases and card games respectively. Learning of multiple concepts with predicate invention has also been addressed in Hyper [20,18]. This system is one of the latest systems using demand-driven predicate invention in robotics domains. Hyper starts with background knowledge and can perform predicate invention adding placeholder predicates (predicates which are being invented) to the target predicates that are being induced. It works automatically once the experimental setup has been properly prepared. The system reuses previously learned concepts for learning new concepts. Positive and negative examples are provided to the system, the positive examples are obtained by the robot while it explores the environment guided by a human (through commands) and the negative examples are synthetically generated. The system was tested for learning the concepts of movable, not movable, and obstacle. In Hyper, multi-predicate and predicate invention is supported, but, predicate invention is performed using templates for new predicates that should be designed before the learning process. Also, examples should be provided to Hyper, by databases or by a manually guided exploration.

ADC introduces new predicates automatically, as SPI, TRP and Hyper, but also automatically discovers examples of potential concepts from the environment. In this sense, ADC does not need databases of examples or using pre-defined templates for the discovery of new predicates. ADC is able to collect positive examples and create negative examples for the induction process. Also as TRP and Hyper, ADC performs incremental hierarchical multi-predicate learning because of its graph-based representation. Although, other approaches have been proposed to also learn hierarchical concepts (e.g., [12,21–27]), they are usually designed to work on databases or in controlled environments, as in SPI, TRP and Hyper.

Recently, Meta-Interpretive Learning (MIL implemented by Metagol$_{D/O}$) [28,29] has been proposed in ILP based on meta-rules, induction and abduction to produce higher-order datalog programs, which takes advantage of predicate invention and recursivity. MIL is based on incremental declarative multi-predicate learning, using meta-rules to conduct the search of the hypothesis from a set of examples. Meta-rules are like templates where the meta-interpreter performs substitutions to introduce new predicates (predicate invention). Predicate invention creates predicates representing relations instead of objects or propositions. Metagol$_D$ has been used to learn high-order concepts in three domains [29]: the East–West train challenge, NELL language learning task and for top-down construction of re-usable robotic strategies in a simplified model of the world (for construction of walls). Metagol$_O$ has also learned general strategies representing a set of plans (as those learned by traditional AI planning) that reduce the use of resources (e.g., battery, distance) [30]. Metagol$_O$ has shown the advantage of using composite objects and actions (formed by other primitive and/or composite objects and actions) to produce resource efficient strategies from examples. Experiments with composite elements were performed with a simulated humanoid robot in one-dimensional space in tasks of delivery and sorting.

ADC is based on inverse entailment induction (Progol [31]) with first-order logic and can perform incremental multi-predicate learning with predicate invention, as Metagol. Metagol has shown the advantages of learning composite objects and actions, meanwhile ADC has been used to discover composite concepts about objects in robotics domains. The use of meta-rules in MIL is a powerful tool for predicate invention, but it depends on the number and design of the meta-rules to produce useful predicates. Also, Metagol, like traditional ILP systems, requires that sets of examples be provided by the user. In this sense, the main advantage of ADC over other ILP systems, including Metagol, is its ability to automatically discover and collect examples of potential concepts directly from the environment. ADC obtains and creates

its positive and negative examples automatically. ADC performs predicate invention without a template/guide for new predicates, and although, recursivity is not yet supported by ADC, it could be supported extending the graph-based representation.

3. Automatic discovery of relational concepts by an incremental graph-based representation through exploration

Our approach, called ADC (for Automatic Discovery of Concepts), is designed to provide an agent with the skills to find potentially relevant concepts while exploring an unknown environment. The main parts of ADC are:

1. Explore: Traverse and sense the environment following an exploration strategy.
2. Represent: Given the information from the sensor's readings verify which predicates of the background knowledge can be satisfied with it. Match each predicate, representing objects or relations of the world, with vertexes and edges to incrementally construct a graph.
3. Induce: After reaching a goal state during the exploration, find approximately equal frequent sub-graphs from the constructed graph and create sets of similar sub-graphs. For each set induce a new concept through an Inductive Logic Programming algorithm, using the corresponding first-order logic representation of the sub-graphs. Add the new concepts to the existing background knowledge.
4. Simplify: Each time a new concept is induced, replace the induced concept in the original graph by a node and repeat the process until no more common sub-graphs can be found, allowing the discovery of hierarchical concepts.

The details about these main parts of the proposed algorithm are presented in the following subsections. Also, the pseudo code of the proposed method is presented in Algorithm 1 and Algorithm 2.

Algorithm 1 The ADC algorithm.

1: Given definitions for objects O and relations among objects R as background knowledge BK, a set of primitive actions A, an exploration strategy E, and a set of conditions to identify target goals T
2: Set groups $Cl = \emptyset$ and graph G = null
3: **repeat**
4: Perform action $a \in A$ to explore the environment using E
5: Capture information from the robot's sensors and identify objects and their relations in the environment using its current BK and add them to graph G, where objects/attributes = nodes, relations = edges
6: **until** until current state satisfies T
7: **while** A frequent sub-graph g is found in G **do**
8: **if** g has *relevant elements/similar structure* to the instances of an existing group $C_i \in Cl$ **then**
9: Let $C_i = \{g\} \cup C_i$
10: **else**
11: Create a new group $C_i = \{g\}$, $Cl = \{C_i\} \cup Cl$
12: **end if**
13: Let $d =$ *Induce Concept* (g, C_i, Cl) and $BK = \{d\} \cup BK$
14: G = Graph G compressed using d (with instantiated variables)
15: **if** G cannot be compressed using d **then**
16: G = Graph G compressed using g
17: **end if**
18: **end while**

3.1. Incremental graph-based representation

During the exploration phase, ADC incrementally builds a graph G representing objects and relations identified from sensor readings in the environment. In order to accomplish this purpose, a set of basic objects O and relations R between objects (e.g., touches, on, near etc.) as well as some basic actions (e.g., move-forward, turn-right, etc.), represented by predicates, are provided by the user as background knowledge BK to explore the environment. So, while the agent is exploring its environment with its current known actions, it tries to identify elements of the environment using its current predicates in the BK. These elements are incrementally added to a graph-based representation of the environment, where objects and their attributes are represented as vertexes and relations as edges. When the arity of a relation is greater than two and there is not direct mapping to a simple graph, we use conceptual graphs [32] to represent objects and their relations as concepts and conceptual relations (both, as vertexes of a graph). In this way, the agent incrementally builds a graph until it reaches an intrinsic goal (defined by a function that considers the current state interesting in some way) or an extrinsic goal (defined normally by the user representing a goal of the current task). This process is illustrated in Fig. 1 and referred in the Algorithm 1 in line 5.

3.2. Induction of potential relational concepts

In this research, the agent is not told what concepts to discover or what constitutes a valid set of examples. To guide the learning process in this research it is assumed that information that is repeated in the environment constitutes a potentially useful concept.

3.2.1. Identification of relational concepts instances in the graph-based representation.

A graph-based representation G allows the system to identify potential concepts by searching for frequent sub-graphs g. Since the identified sub-graphs are structures (formed by objects and their attributes and relations among them) found many times in the environment representation, they may represent relevant structures of the environment and could be used by the agent to characterize it. Characterizing the environment with new concepts can be useful to identify different states of the world and solve tasks involving state changes.

We discover frequent sub-graphs which best compress an input graph using an inexact matching measure and the minimum description length principle (MDL) by the Subdue [21] system. Subdue performs a computationally constrained beam search in order to discover substructures. At the beginning, a substructure is selected matching a single vertex of the graph, then, a series of expansions are performed iteratively through all possible directions (edges) to create new substructures. On each iteration, the substructure which best compresses the current graph is selected, guided by MDL, among all (or up to a computational limit) possible substructures generated by expansions. Additional background knowledge and pruning methods are also used during the search of substructures [33]. The similarity among substructures, found by Subdue, is calculated by an inexact matching measure that depends on a threshold value, which measures the fraction of the size (vertexes and edges) by which graphs can differ. A value of 0 (default value) implies that the graphs must match exactly. For our purpose, this value is set low to promote finding similar (with small differences) instances of the same concept, and deal with possible errors or noise in the sensors' information. Although finding sub-graphs is equivalent to sub-graph isomorphism, which is NP-complete [34],

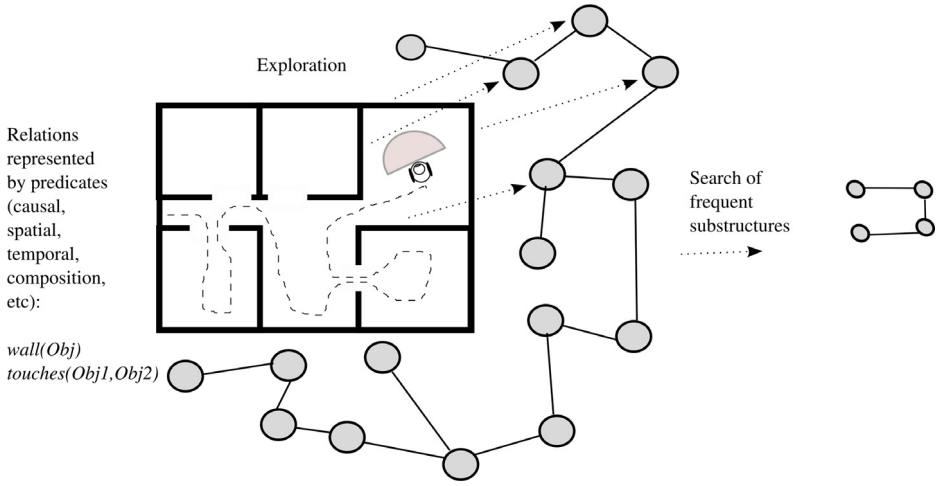

Fig. 1. A robot explores the floor of a building, using an initial background knowledge, identifies basic elements and builds a graph from which frequent substructures are identified as instances of potential concepts.

Table 1
Examples of CPU time spent by Subdue (release 2011 available at [35]) in substructure discovery for different sizes of graphs. Parameters of testing: beam width: 4, evaluation method: MDL, directed/undirected edges, minimum size of substructures: 1, maximum size of substructures: 2500, allow overlapping instances: false, prune: false, threshold: 0.1/0.3. OS: Debian Linux 7.0. Hardware: CPU: AMD Sempron 3600+ 2 GHz, RAM: 4 Gb 667 MHz.

Number of vertexes	Number of edges	CPU time (s)
20–50	20–60	≈0.07
51–80	70–80	≈0.18
81–135	81–140	≈5.15
...	...	...
4500	4000	≈15.05

Subdue is designed to find a large number of substructures under computational constraints. The inexact matching process for a graph 1 with n vertexes and a graph 2 with m vertexes, where $m \geq n$, has a complexity of $\mathcal{O}(n^{m+1})$, although its performance can be improved by the use of a branch-and-bound search algorithm [33]. Some examples of Subdue performance, with the parameters values used in this research, are presented in Table 1. This process of sub-graph discovery is performed to find each *frequent subgraph* g from G referred in Algorithm 1 in line 7.

The performance of the ADC algorithm using the sub-graph discovery method described above depends on its background knowledge (number of objects and relations), the characteristics of the environment, and the size of the graphs formed during the exploration of an environment. It is expected that with large background knowledge and environments, the performance of ADC can be seriously affected. In the experiments presented in this paper with a simulated robot the performance of ADC was in terms of seconds. ADC can only discover concepts derived from frequent sub-graphs (with at least one relation among their objects). It is currently limited to represent them as non-recursive first-order clausal definitions without functional symbols or negation. Using more expressive definitions is left as future work.

3.2.2. Grouping relational concepts instances in sets

We identify a very large number of sub-graphs by Subdue, many of which are redundant or instances of more general concepts. In order to learn more general concepts, ADC performs a generalization process over graphs. ADC creates sets of similar instances of sub-graphs. Each group has all the similar sub-graphs, all their equivalent clausal form, and the current generalization of the clauses. When a new group is created with a single graph, its clausal definition also corresponds to the current generalization of the group.

When a sub-graph g_i is discovered, the sub-graph is kept but also it is transformed into a clause cl_{g_i}. The body of the clause cl_{g_i} is constructed with all the elements in the sub-graph g_i, each object $o_n \in O$ (or concept c) and relation $r_n \in R$ of g_i correspond to a literal l_n of the body of the clause. The head of the clause is defined with a new predicate name with its arguments formed by the distinctive arguments existing in the literals in the body of the clause. For instance, in Fig. 2 a sub-graph is discovered in a graph of a scene with pieces of furniture. This sub-graph is formed by four objects (one *seat* and three *legs*), one attribute of an object (*flat* with the link *is*), and three relations (*on*). When this sub-graph is represented by a clause, each object and relation is represented by a literal. The arguments of the literals for the objects are the name of the instances (*seat*1, *leg*1, *leg*2, *leg*3) and the attributes of the objects (*flat* in the case of *seat*). The arguments of the literals of the relations are the objects involved in that relation. The arguments of the head are the arguments of the literals in the body of the clause (*seat*1, *flat*, *leg*1, *leg*2, *leg*3). As we produce clauses from sub-graphs, we can take advantage of powerful machine learning methods for relational concepts. Also, we can exploit the expressiveness and understandability of a first-order language (e.g., include variables and relations in the definitions). The clauses induced by ADC can be directly used by a robot as independent pieces of knowledge, and can participate in the discovery of knowledge.

In many cases, different instances of the same concept have a similar structure with almost the same elements, however, this is not always the case for some concepts, as illustrated in Fig. 3. To address this issue, we introduce two measures to define when to include an instance to a group. The first measure considers the similarity between the type of elements (objects and relations) involved in an instance, while the second one measures the similarity in terms of structural matching. Both measures are applied to the clausal definitions to create the groups. Suppose we have the six instances shown in Fig. 3. If we try to group those six sub-graphs measuring only their structural similarity, it is likely that sub-graphs 4 and 5 will be placed in a group, sub-graphs 2 and 6 in other group, and possibly sub-graphs 3, 4 and 5 will be forming another group. If we consider their basic elements, sub-graphs 1–4 could be placed in one group (with common elements: seat S, leg L), and sub-graphs 5 and 6 could be in another group (with common elements: flat board F, leg L). We used both similarity measures with preference to the measure involving common elements. If a

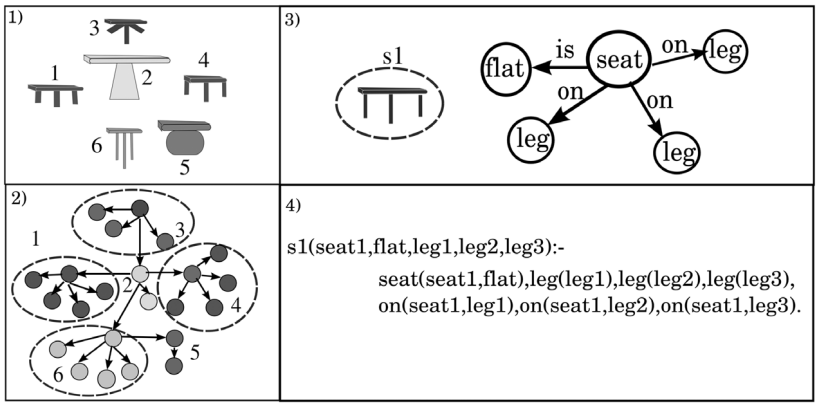

Fig. 2. A scene with six pieces of furniture (1) is represented by a graph (2), from which a common sub-graph is obtained (3) and represented by a Horn clause (4).

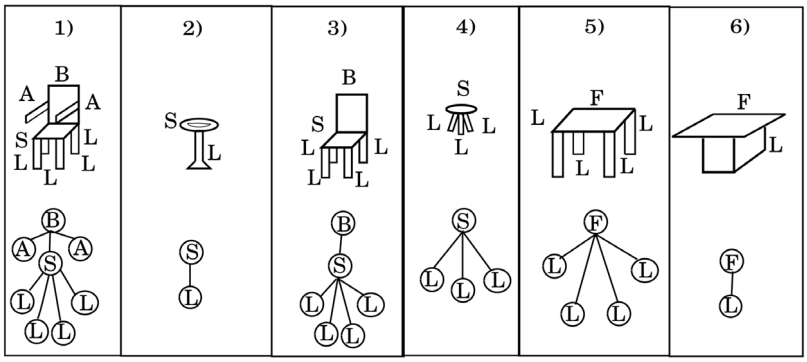

Fig. 3. Examples of concepts (chairs, stools and tables) with their graph representations. Graphs 1–4 illustrate chairs and stools, while graphs 5 and 6 illustrate two kinds of tables. This figure shows how instances of different concepts can have similar structures but different elements (e.g., figures 2, 6, 4 and 5), and how instances of the same concepts can have similar structures but different elements (e.g. figures 1, 2, 5 and 6).

new instance is similar to a group in terms of common elements it is added to that group, otherwise the measure in terms of structural differences is used. If there is no group satisfying a threshold value for at least one of the two measures, then a new group is created with the new instance.

With the first measure a sub-graph g_i is added to a group C_k if certain percentage of the objects and relations of the clause cl_{g_i} of g_i can also be found in the common objects and relations R_k found in the instances of that group. This measure allows to cluster sub-graphs which share the same objects and relations although their structures and sizes could be different (this measure is used in Algorithm 1 line 8).

Let cl_{g_i} be a new instance clause and L_g be its set of literals. Let C_k be a group of instances and R_k be the set of common literals in the clauses of the instances in C_k according to Eq. (3). A sub-graph g_i is added to a group C_k, if a large proportion of the literals L_g in cl_{g_i} are common to R_k (see Eq. (1)) and if a large proportion of the most common literals in C_k (i.e., R_k) are common to the literals in cl_{g_i} (see Eq. (2)).

$$g_i \in C_k \quad \text{iff} \quad \frac{|L_g \cap R_k|}{|L_g|} \geq th_1 \tag{1}$$

$$g_i \in C_k \quad \text{iff} \quad \frac{|L_g \cap R_k|}{|R_k|} \geq th_2 \tag{2}$$

$$l \in R_k \quad \text{iff} \quad \sum_{i=1}^{S_{C_k}} |\{l\} \cap L_{g_i}| \geq \frac{\sum_{j=1}^{|L_{C_k}|} \sum_{i=1}^{S_{C_k}} |\{l_j\} \cap L_{g_i}|}{|L_{C_k}|} \tag{3}$$

where S_{C_k} = number of sub-graphs of C_k, L_{C_k} = set of literals of all graphs in C_k and th_1, th_2 = threshold values. Let $N(l)$ = name of literal l and L a set of literals:

$$\{l\} \cap L = \begin{cases} l & \text{if } N(l) = N(k) \text{ and } k \in L \\ \emptyset & \text{otherwise.} \end{cases} \tag{4}$$

The second measure takes into account the size and structure of the sub-graphs. A sub-graph g_i is added to a group of sub-graphs C_k if the average cost of structural changes to make $g_i = g_k \in C_k$ for all the sub-graphs in C_k is smaller than a threshold value (see Eq. (5)). The $matchcost(g_i, g_k)$ corresponds to the inexact matching measure used in Subdue, this function calculates the minimum cost to convert one graph to another, considering a cost based on deletion, insertion and substitution of vertexes and edges, through a branch-and-bound tree search algorithm [21] (this measure is used in Algorithm 1 line 8).

$$g_i \in C_k \quad \text{iff} \quad \frac{\sum_{j=1}^{S_{C_k}} matchcost(g_i, g_j)}{S_{C_k}} \leq th_3 \tag{5}$$

where th_3 = threshold value.

The similarity between instances in a group depends of the values given to the thresholds $th1$, $th2$ and $th3$.

If the thresholds $th1$ and $th2$ are set to 0, the instances in a group could have clauses without any literal in common. If these thresholds are set to 1, all the instances in a group should have the same literals in their clauses. Thresholds $th1$ and $th2$ higher than 0.5 promote that instances, with a large number of literals in common in their clauses, are grouped together. In our experiments these thresholds values were empirically set to $th1 = 0.6$ and $th2 = 0.7$ for all the domains. Varying the threshold $th3$ produces instances with more or less similar structures (vertexes and edges)

in the same group. If it is set to 0, instances in a group will have the same structure. In our experiments, the best results were obtained when this value was less or equal than 3.0 due to the size of the sub-graphs (less than 30 vertexes) and the cost of comparing sub-graphs by the inexact matching measure. Higher values of $th3$ may be needed for large graphs. Instances with very different elements and/or structures in the same group produce very general definitions, possibly covering several useful concepts.

3.2.3. Induction of definitions of potential relational concepts

Each time a sub-graph is added to a group, ADC runs a generalization process (see the Algorithm 2). As we mentioned, each sub-graph g_j in the group C_k is also represented by a clause cl_{g_j}, then, when a new discovered sub-graph g_i is added to a group C_k, its respective clause cl_{g_i} together with the rest of the clauses of the instances in the group are used as positive examples Ex^+ for the induction of a concept definition *newD*.

Also a set of negative examples Ex^- is constructed. This set includes clauses of the instances cl_{g_o} and concept definitions d_j of other groups $C_j \in Cl$ and negative examples artificially created (Ex^-_{ar}) by deleting literals from positive examples Ex^+ of the group where the new graph was added (see Eq. (8)). This prevents ADC from over-generalizations from few examples. The group concept definition is re-induced each time a new sub-graph is added to the group (see Algorithm 2). Currently, the induction process is performed by the Progol system based on inverse entailment and a general-to-specific search [31]. The process is described as follows:

$$newD \leftarrow Induce(BK, Ex^+, Ex^-) \quad \% \text{ induces a new definition} \tag{6}$$

where:

$$Ex^+ = \{cl_{g_m} \in C_k\} \quad \% \text{ set of positive examples} \tag{7}$$

$$Ex^- = \{d_j \in BK | d_j \neq d\} \cup \{cl_{g_o} \in C_j | C_j \in Cl \wedge C_j \neq C_k\} \cup Ex^-_{ar} \quad \% \text{ set of negative examples} \tag{8}$$

where d is the current concept definition in the group. If the group has one instance, this instance will be the current definition. If the group has two or more instances, their induced generalization will be the current definition. And,

$$Ex^-_{ar} = \text{Modify}(Ex^+) \tag{9}$$

where, *Modify* creates new negative examples from the positive examples Ex^+ (line 5 in Algorithm 2). Each $cl_g \in Ex^+$ has the form $h_g \leftarrow O, R$ where h_g is the head of the clause, $O = \{o_1, o_2, \ldots, o_n\}$ is the set of literals in the clause that describes objects, and $R = \{r_1, r_2, \ldots, r_m\}$ is the set of literals that describes relations among objects. *Modify* creates the following clauses as negative examples (if they are not members of Ex^+):

$h_g \leftarrow O$ (a clause without relations)
$h_g \leftarrow R$ (a clause without explicit objects)
$h_g \leftarrow O, R' | R' \subset R$ (clauses with a smaller number of relations).

Suppose we have the clause cl_{g_l} as one positive example:

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3), on(seat1, leg1), on(seat1, leg2), on(seat1, leg3)$$

and

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3), on(seat1, leg1), on(seat1, leg2)$$

as a second positive example cl_{g_m}. Both clauses as instances of the same group. Then, the negative examples artificially created from the first positive example cl_{g_l} are:

Clause without relations:

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3).$$

Clause without explicit objects:

$$s1(seat1, flat1, leg1, leg2, leg3) :- on(seat1, leg1), on(seat1, leg2), on(seat1, leg3).$$

Clauses with a smaller number of relations:

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3), on(seat1, leg2), on(seat1, leg3).$$

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3), on(seat1, leg1).$$

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3), on(seat1, leg1), on(seat1, leg3).$$

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3), on(seat1, leg2).$$

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3), on(seat1, leg3).$$

Note that the clause:

$$s1(seat1, flat1, leg1, leg2, leg3) :- seat(seat1, flat1), leg(leg1), leg(leg2), leg(leg3), on(seat1, leg1), on(seat1, leg2)$$

was not produced as a negative example as it is one of the positive examples (cl_{g_m}) of the group.

The artificial negative examples (Ex^-_{ar}) are created to avoid over-generalization. At the beginning, each group has few instances of a potential concept and it is likely that the induced concept will be too specific. However, as different instances are added to the group, its definition becomes more general, as the artificial negative examples (clauses) are produced only if they cannot be found in the positive examples. Since, it is unknown how many instances will be collected in each group and what elements (objects and relations) will have future instances, the automatic generation of negative examples helps us to reduce possible over-generalizations. Also, the Ex^-_{ar} are useful when positive examples of some groups cannot be used as negative examples to other groups, because there are significantly differences between groups.

The number of concepts induced by ADC can grow quickly and several strategies can be used to control it. In particular, the concepts can be arranged in lattices that can be used to decide which concepts to consider at the same time. The number of concepts which are learned in a session of learning can be limited, and also, methods to select relevant concepts can be developed. In this work we have made some experiments with the use of lattices.

3.3. Hierarchical relational concepts

The new induced definition *newD* can be used to compress the current graph G by substituting the instances of the whole sub-graph (representing *newD*) by simple nodes with their corresponding arcs. If the new definition has variables, they must be instantiated before the concept can be used for compressing the graph. In our case, the variable arguments of the predicates are arbitrarily, although consistently, instantiated. The definition, with

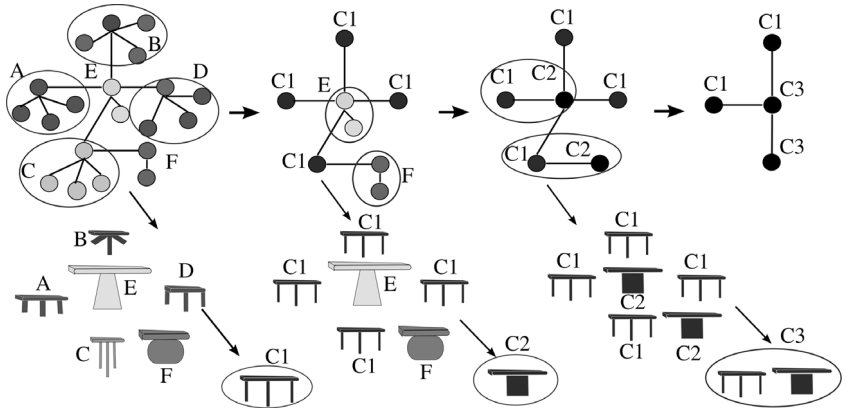

Fig. 4. Example of compression of a graph and generation of a new input graph. In the figure, a structure representing four frequent sub-graphs is used to compress the graph and create the concept of stool (C1). Then, the process is repeated and the concept of table is generated (C2). Finally, these concepts are used, in the compressed graph, to create the concept of "table and stool" (C3).

Algorithm 2 Induce definitions of potential concept

1: *Induce Concept* (g, C, Cl)
2: Let d_1 = clause definition constructed from g
3: **if** group C has associated a definition d in BK **then**
4: Let $Ex^+ = C$
5: Create artificial negative examples guided by current instances in Ex^+, $Ex^-_{ar} = Modify(Ex^+)$
6: Let $Ex^- = \{d_i \in BK | d_i \neq d\} \cup \{g_o \in C_i | C_i \in Cl \wedge C_i \neq C\} \cup Ex^-_{ar}$
7: $newD = Induce(BK, Ex^+, Ex^-)$
8: **if** $newD$ could not be induced **then**
9: Discard $newD$
10: Let $C = C - \{g\}$
11: Create a new group $C_j = \{g\}$, $Cl = \{C_j\} \cup Cl$
12: Set $d = d_1$
13: **else**
14: Set $d = newD$
15: **end if**
16: **else**
17: Set $d = d_1$
18: **end if**
19: Return d

its original arguments (variables), however, is kept and can be used for further learning. Currently, the compression of G using $newD$ is performed through the algorithm provided by Subdue, applying the inexact matching measure to find the sub-graph representing $newD$ in G [21]. The compressed graph is then used as new input to our algorithm to find new common sub-graphs, from which new, hierarchical concepts can be induced. Since the new compressed graph has vertexes that represent previously learned concepts, when new frequent sub-graphs are found on it, it is possible that these new common sub-graphs include learned concepts among its vertexes (see Fig. 4).

4. Experiments

ADC was applied in three domains: floors, polygons and furniture, to learn concepts that could be used for manipulation and navigation tasks. First, we present the results of the experiments obtained by ADC in these domains, with some illustrations of the kind of concepts learned by the system. We then present an evaluation of these learned concepts in terms of how well they are recognized by users and how useful they are in new environments. The thresholds for the algorithm were set to $th1 = 0.6$, $th2 = 0.7$ and $th3 = 3.0$. The threshold

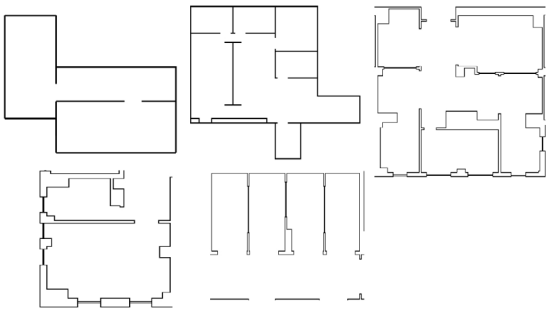

Fig. 5. Floor maps used by the robot to learn "structural" concepts.

for similar substructure search by Subdue was set to 0.3 for the floors and 0.1 for the polygons and furniture domains. The original implementation of Subdue (release 2011 available in [35]) was used in the experiments. The background knowledge in these experiments did not include numerical data, however, the proposed algorithm can support numerical data if needed. In [36] can be found the background knowledge used in these experiments by ADC.

4.1. Floors

The first experiments were performed on simulation using *Player/Stage* [37] with a *Pioneer* 2 robot equipped with a laser sensor (with a range of 180°). In this domain, the robot explored five maps of floors of buildings, shown in Fig. 5. The robot built five graphs with 24, 38, 135, 45, and 50 nodes for each map, respectively. Different strategies can be used to explore the environment. They can be guided by unexplored areas, using an intrinsically motivated reward function or be guided by the user. In these experiments, the user guided the robot to cover the whole floors once.

The initial background knowledge provided to the robot consisted of two predicates, *wall* and *touches* (see Table 2 and Fig. 6). An incremental algorithm was used for the identification of lines from laser readings [38], from which instances of walls were identified.

In this first set of experiments we contrasted the results obtained with ADC with those that can be obtained by Subdue, a graph-based discovery algorithm. Many induction systems, such as Subdue, cannot be used in an incremental way or significantly decrease their performance if the data is given incrementally. Table 3 shows the number of concepts induced by ADC and by

Table 2
Background knowledge provided to learn concepts about floors.

BK	Name	Description
Object	wall/1	Representing a wall/line segment in the environment, where its argument is an identifier (constant or variable) of an instance.
Relation	touches/2	Representing two walls forming a corner, where its arguments are identifiers (constants or variables) of two instances (wall, concept).

Table 3
Results from the floors. In ADC the graphs were incrementally constructed from exploring the environment. In Subdue the complete graphs were given either as a single graph or sequentially.

Floors	Group with singleton	Group with multiple subGs	Hierarchical concept	Total
ADC	2	2	12	16
Subdue: ($G1 \cup \cdots \cup G5$)	3	–	20	23
Subdue: (G1, …, G5)	12	–	30	42

Subdue, if graphs are given sequentially (as in ADC) or when all the graphs are given at the same time as a single graph. The second column, *Group with singleton*, refers to groups with a single sub-graph, *Group with multiple subGs*, refers to groups with multiple sub-graphs, and *Hierarchical Concept*, refers to concept definitions that include previously learned concepts.

In Subdue, if the graphs are given sequentially, as in our learning setting, a large number of isolated sub-graphs are induced, many of which represent equivalent concepts. This behavior is reduced, to a certain extent, when all the graphs are given to Subdue at the same time as a single graph, as it is able to join some equivalent sub-graphs from different graphs. Nevertheless, it still suffers from creating too specific concepts.

Among the concepts learned by ADC, a user could identify concepts such as "room" (C1 in the figure) and "set of rooms" (C4 in the figure) among others, illustrated in Fig. 7. The definitions of these discovered concepts are:

c1(A,B,C,D,E):- wall(A),wall(B),wall(C),wall(D),wall(E),
touches(A,B),touches(B,C),touches(C,D),touches(D,E).
c4(A,B,C,D):- c1(A),c1(B),c1(C),c1(D),touches(A,B),
touches(B,C),touches(C,D).

The interpretation and successful application of these concepts depends on the way that the basic elements (objects and their relations) are identified by the robot in an environment. In this domain, the robot was able to measure its orientation, coordinates and distance to other objects to identify correctly the different elements of the definitions given. For example, although the definition in $c1$ does not include an explicit verification to ensure that the walls are different, during the exploration, the robot uses its sensors to identify five different instances of walls as input to the concept $c1$.

4.2. Polygons

In this case, we artificially created five graphs, shown in Fig. 8, to learn different concepts related with polygons. The number of nodes in the graphs is 26, 55, 41, 61, and 56, respectively. The predicates used as background knowledge are line, curve and two relations representing concave and convex angles (see Table 4 and Fig. 9).

ADC was able to discover five general concepts: "triangle", "pentagon", "square", "hexagon" and "irregular polygon with a concave angle". The remainder of the learned concepts had these simple concepts among their elements. The total number of concepts induced by ADC is given in Table 5, some of which are illustrated in Fig. 10.

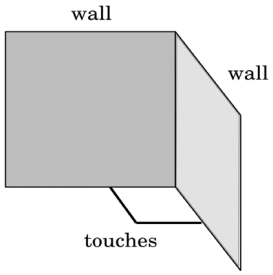

Fig. 6. This scene illustrate the background knowledge used in the floors domain.

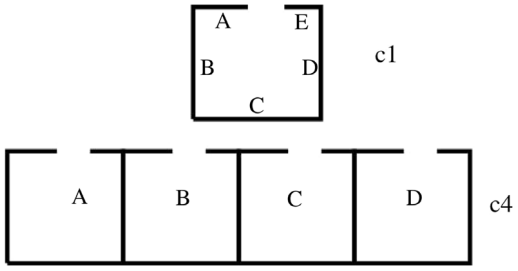

Fig. 7. Examples of induced concepts in the floors.

4.3. Furniture

In the last experiment, we used four graphs based on a real "Reading and Internet" room, shown in Fig. 11. The number of nodes in the graphs is 77, 70, 31, and 22, respectively.

The predicates provided as background knowledge (see Table 6 and Fig. 12) represent several objects that can be identified in rooms, like flat surfaces, legs, lamps, etc., and relations among them, like next-to, above, etc.

Table 5 presents the number of concepts discovered by ADC. Again, they are arranged as groups with singletons, groups with several sub-graphs, and hierarchical concepts. Some of the

Table 4
Background knowledge provided to learn concepts about polygons.

BK	Name	Description
Object	line/1	Representing a line segment.
	curve/1	Representing a curve segment.
		Each object has one argument that is an identifier (constant or variable) of an instance.
Relation	angcc$N$/2	A relation between two elements forming an angle greater than 180°, where N is the angle.
	angcx$N$/2	A relation between two elements forming an angle less than 180°, where N is the angle.
		Each relation has two arguments that are identifiers (constants or variables) of two instances (line, curve or concept).

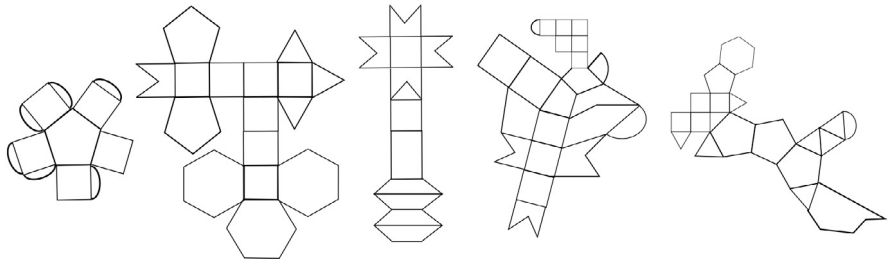

Fig. 8. Five figures used in the first domain, in the learning of concepts about polygons, are illustrated.

Table 5
Concepts learned by ADC in the polygons and the furniture domains.

Domain	Group with singleton	Group with multiple subGs	Hierarchical concept	Total
Polygons	5	5	24	34
Furniture	1	5	18	24

concepts discovered by ADC were: "table" (a flat board on four legs), "footstool" (a footstool on four legs), "chair" (back and seat on four legs), "lamp on table" (lampshade on light bulb base on a table), "chair in front of table", and "table next to table". The remainder of the concepts learned, as well as in the previous domain, had simple concepts among their elements. Some examples are illustrated in Fig. 13. In this domain the concept of "chair" also includes "armchair" and "sofa".

The learned concepts, in the three domains, could be used by a robot to characterize states of the world. In the first domain, it could help a robot to reason, for instance, in terms of rooms rather than corners. These kinds of concepts could be useful in tasks such as navigation to identify parts of an environment. In the second domain, the concepts can be useful given that geometric figures are commonly encountered in robotic applications (e.g. classification of objects, navigation, reconstruction of environments, among others). The concepts learned in the third domain to identify common furniture can also be useful for service robots.

4.4. Evaluation of concepts

So far, we have shown that ADC is capable of inducing several concepts, from different domains, and that the learned concepts can be more general than those produced by a graph-based discovery system like Subdue, even is all the graphs are given simultaneously. However, it is not clear how useful or intuitive are these concepts. Evaluation is not easy for automatic discovery systems. ADC automatically induces several concepts while exploring its environment and some of them may not correspond to what the user is expecting. In this section we evaluate the concepts discovered by ADC following two strategies:

1. Compare the concepts discovered by ADC with those concepts expected by independent users.
2. See if the concepts discovered by ADC can be effectively used in new environments which are labeled by independent users.

We performed these experiments in the three domains: floors, polygons and furniture.

Ten independent non-expert users, without computer science knowledge, and one independent expert user, with computer science knowledge, identified concepts in the different domains to compare them with those discovered by ADC. They also evaluated the concepts learned by ADC. These comparisons are presented below.

The new floor maps and the respective labels given by the expert user are shown in Fig. 14. As can be seen, the expert user identified 29 structures of two concepts (hallways and rooms).

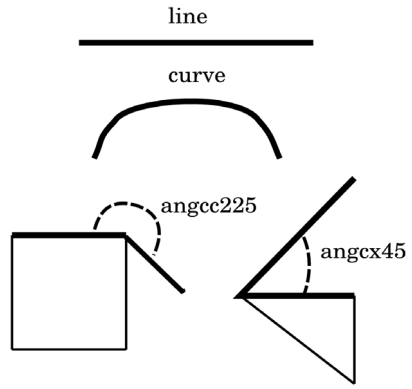

Fig. 9. This scene illustrate the background knowledge used in the polygons domain.

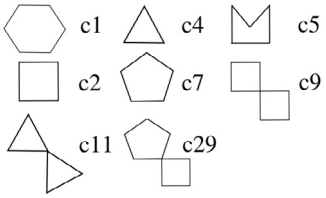

Fig. 10. Examples of induced concepts in polygons.

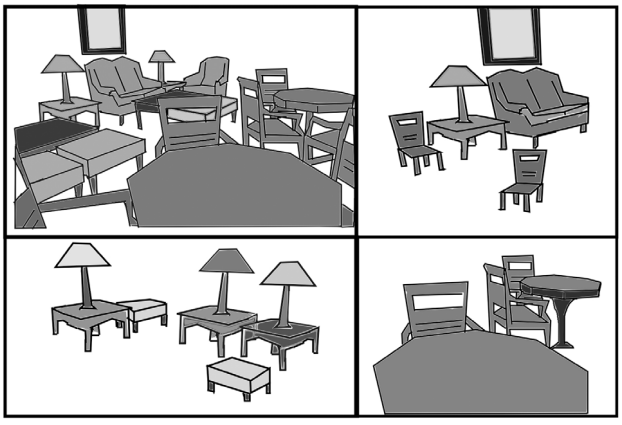

Fig. 11. Drawings based on a real "Internet and Reading" room used as basis for create four graphs of the furniture domain.

The concepts discovered by ADC for polygons were tested in Fig. 15. The expert user identified 16 objects in the new graph and gave 43 labels.

The scene used to construct a graph and evaluate the concepts of furniture learned from ADC is shown in Fig. 16. The expert user identified 16 objects and produced 27 labels of simple objects

Table 6
Background knowledge provided to learn concepts about furniture.

BK	Name	Description
Object	flat_board/1	Top of a table.
	leg/1	Legs as those present in tables, chairs, sofa, armchair or footstool.
	light_bulb_base/1	Bottom of a lamp.
	lampshade/1	Top of a lamp.
	framed_picture/1	A framed picture.
	footrest/1	Top of a footstool.
	seat/1	A seat for a chair, sofa or armchair.
	back/1	The back of a sofa, armchair or chair.
	arm_support/1	A support for arms in armchairs and sofas.
		Each object has one argument that is an identifier (constant or variable) of an instance.
Relation	On/2	When one object is on another object.
	next_to/2	When one object is next to another object.
	behind/2	When one object is behind another object.
	in_front_of/2	When one object is in front of another object.
	above/2	When one object is above another object.
		All relations have two arguments that are identifiers (constants or variables) of two instances (simple objects of furniture or concepts). All relations are absolute among objects.

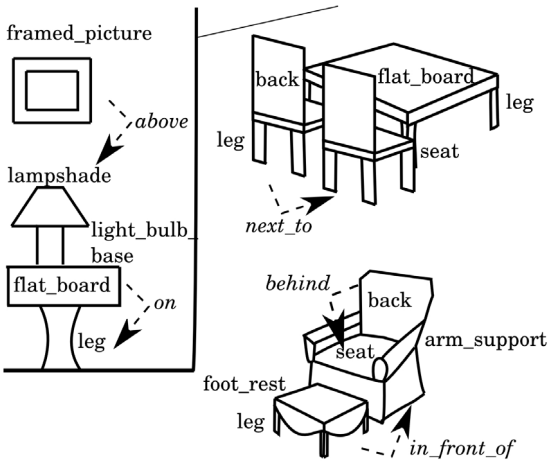

Fig. 12. This scene illustrate the background knowledge used in the furniture domain.

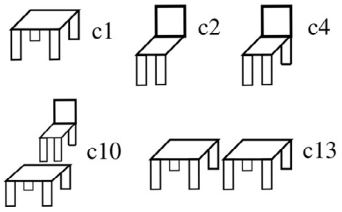

Fig. 13. Examples of induced concepts in the furniture domain.

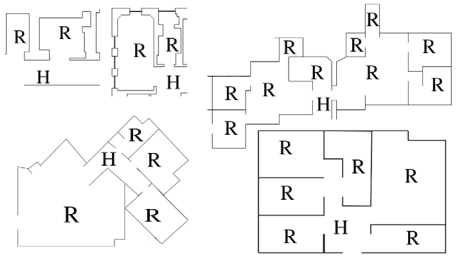

Fig. 14. Unknown floor maps used to evaluate the concepts discovered by ADC with those labeled by an independent expert user. Labels of different sections of the environments given by an expert user are shown in the image with the letters R and H (Room or Hallway respectively).

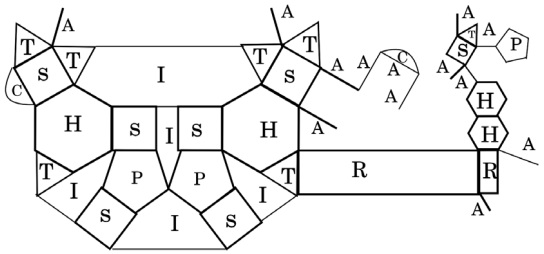

Fig. 15. The figure used to evaluate the learned concepts of polygons is illustrated. A graph was formed describing the basic figures in this scene. Labels of different figures given by the expert user are shown in the image (T = *triangle*, R = *rectangle*, S = *square*, H = *hexagon*, P = *pentagon*, I = *irregular polygon*, C = *curve*, A = *angle*).

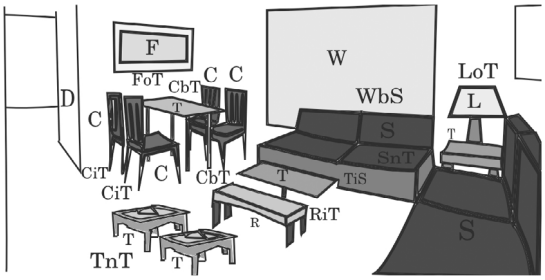

Fig. 16. The scene of a real living/dinner room used to evaluate the concepts learned about furniture is illustrated. A graph was formed describing the basic relations in this scene. Labels of different elements of furniture and their relations given by the expert user are shown in the image (*Objects* : F = *framed picture*, D = *door*, C = *chair*, L = *lamp*, T = *table*, W = *window*, R = *foot rest*, S = *sofa*; *Relations* : b = *behind*, o = *on*, i = *in front of*, n = *next to* (*e.g. WbS* = *window behind sofa*)).

(e.g., chair, table, lamp, etc.) and basic relations among the objects (e.g., The lamp is on the table) for this scene.

Table 7 shows the number of different concepts discovered by ADC in the original graphs divided by the number of concepts identified by the expert user (second column), the number of common concepts between ADC and the expert user (third column), and the concepts discovered by ADC that were validated by the expert user (last column). It can be seen that ADC identified half of the expected concepts of the expert user in two domains and about one third in the other domain. Also, when the expert user was presented with the concepts discovered by ADC, s/he was able to validate a large proportion of them. For example, for the floors

Table 7
Comparison of concepts. The table shows the number of concepts induced by ADC and identified by an independent expert user (2nd. column), the number of common concepts (3th. column), and the number of concepts induced by ADC that were easily identified by the independent expert user (last column).

Domain	ADC/user	ADC ∩ user	ADC concepts validated by user
Floors	8.00 (16/2)	1 (1/2)	6
Polygons	2.13 (34/16)	5 (5/16)	15
Furniture	4.00 (24/6)	3 (3/6)	12

Table 8
Comparison of concepts. In this case an average of ten users results is compared with the results obtained by ADC. The table shows the number of concepts induced by ADC and identified by the independent non-expert users (2nd. column), the number of common concepts (3th. column), and the number of concepts induced by ADC that were easily identified by the independent non-expert users (last column).

Domain	ADC/users	ADC ∩ users	ADC concepts validated by users
Floors	8.00 (16/2)	1 (1/2)	3
Polygons	4.25 (34/8)	5 (5/8)	5
Furniture	3.43 (24/7)	4 (4/7)	4

domain (second row in Table 7), ADC induced 16 concepts and the expert user identified 2 (*room* and *hallway*) in the same graphs. The ratio 8.0 (16/2) (in the second column in Table 7) indicates that ADC discovered eight times more concepts than the expert user ($16/2 = 8.0$). The expert user labeled 2 concepts (*room* and *hallway*) in the graph, but among the 16 concepts induced by ADC in the same graph, only one, the *room* was found in common (1 of 2, in the third column of the Table 7: 1 (1/2)). ADC was not able to discover *hallway* due to the way the graphs are constructed (i.e., it is not able to identify an empty space), but it discovered the concept of *room*, among others. When the expert user was presented with the 16 concepts discovered by ADC s/he decided that 6 of them were valid/useful concepts (the 6 in the fourth column in Table 7).

In a similar way, the rest of users (ten) performed similar analysis in the different domains. Their labeled figures of each domain (floors, polygons and furniture) are shown in Figs. 17–19. The average results (common concepts identified by the majority of users) compared with ADC are shown in Table 8. Here, the majority was defined as at least half plus one, so the user concepts compared with ADC were those identified by at least six users. They were able to identify a similar number of concepts as the expert user did, except in one domain (polygons). However, these users identified/validated less concepts learned by ADC in the three domains. Although a basic explanation of the concept representation and structure was provided to all users, these second results may be due to their lack of knowledge of a first-order language (the users were given the clausal definitions of the concepts) and the presence of complex hierarchical concept definitions.

From these experiments we can conclude that ADC is able to automatically discover concepts that could in principle be used by a robot and that are identifiable by independent users. This opens the possibility for the development of incremental learning systems for robots that require little user intervention. ADC can also be used to automatically discover new concepts, or at least concepts that did not come into the mind of the user, but that may be useful to a user. It should be noted that ADC also discovers concepts without any apparent utility.

We also tested if the concepts learned by ADC could be used to identify them in new environments labeled by independent non-expert users using their own expected concepts. Tables 9 and 10 show the results. It compares the discovered concepts identified by ADC with the labels assigned by the users.

Table 9
Precision, recall, and a comparison among the number of concepts identified by ADC in new instances and the concepts labeled by an independent expert user for the three domains.

Domain	Precision	Recall	ADC/user	ADC ∩ user
Floors	0.95 (19/20)	0.79 (19/24)	0.66 (19/29)	1 (1/2)
Polygons	0.95 (22/23)	0.95 (22/23)	0.51 (22/43)	5 (5/14)
Furniture	0.71 (10/14)	0.59 (10/17)	0.37 (10/27)	6 (6/13)

Table 10
Precision, recall, and a comparison among the number of concepts identified by ADC in new instances and the concepts labeled by independent non-expert users for the three domains.

Domain	Precision	Recall	ADC/user	ADC ∩ user
Floors	0.85 (17/20)	0.65 (17/26)	0.44 (17/39)	1 (1/2)
Polygons	0.91 (21/23)	0.75 (21/28)	0.66 (21/32)	6 (6/7)
Furniture	0.79 (11/14)	0.92 (11/12)	0.58 (11/19)	4 (4/7)

For example, in the first Table 9, in the new floors the expert user produced 29 labels, out of which 24 were labeled as rooms (which is the common concept between ADC and the expert user). ADC labeled 20 spaces as rooms, out of which 19 were labeled by the expert user as rooms (precision) and identified 19 of the 24 rooms labeled by the expert user (recall). The fourth column shows the number of common concepts identified by ADC divided by the number of labels produced by the expert user and the fifth column shows the common concepts between ADC and the expert user that were used to label the environments. The second table shows similar results but comparing the average opinion of the ten independent non-expert users against the ADC results. In this second table, precision and recall were lower than those obtained by the expert user. In this case, just one domain (furniture) obtained better results when compared to the first evaluation of concepts.

Tables A.11–A.13 show the definitions of concepts learned by ADC which were identified by the expert and non-expert users. The label given to each concept is also provided. According to the results, the representation of hierarchical concepts was more difficult to understand by the non-expert users. All users identified additional concepts in the outputs from ADC from those originally identified in the three domains. Some concepts were not explicitly illustrated in the scenes given to the users. For example, the expert user labeled a concept learned by ADC as *stool*, where there were no stools illustrated in the furniture scenes. In the floors, the users identified additional concepts including more than one *room* (see Table A.11: *c4,c8,c9,c15*). In the polygons, additional concepts including more than one polygon and elements of polygons (see Table A.12: *c9,c11,c19–c32*, e.g., *two triangles, two squares, angles*) were identified among the discovered concepts of ADC by the users. In the furniture, some additional concepts with elements of furniture and concepts with more than one piece of furniture involved were identified by the users (see Table A.13: *c2,c7–c16*, e.g. *stool or chair of two legs*).

ADC was also able to recognize with high precision and recall the concepts that were common with the different users in new environments.

5. Conclusions and future work

Robots are becoming increasingly popular and it will be desirable to provide them with more autonomy. A key component will be to enable them to learn by themselves new concepts that could be used to perform their tasks. Automatic concept discovery has been a difficult task in machine learning. In this paper,

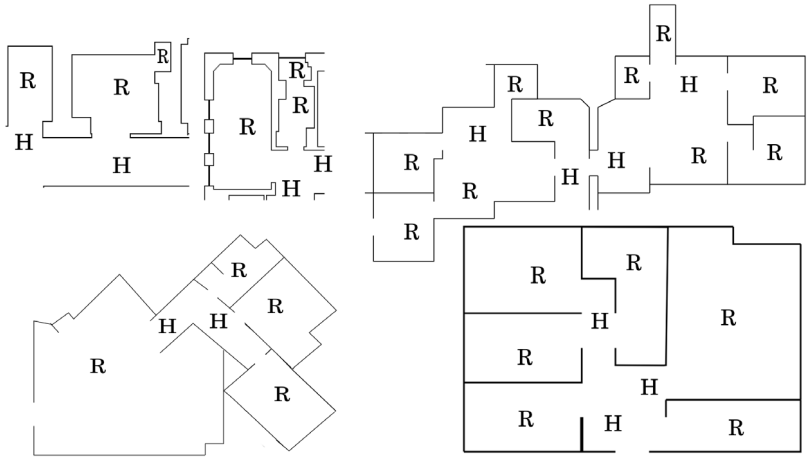

Fig. 17. Unknown floor maps used to evaluate the concepts discovered by ADC with those labeled by 10 independent non-expert users. Labels of different sections of the environments given by the users are shown in the image with the letters R and H (Room or Hallway respectively).

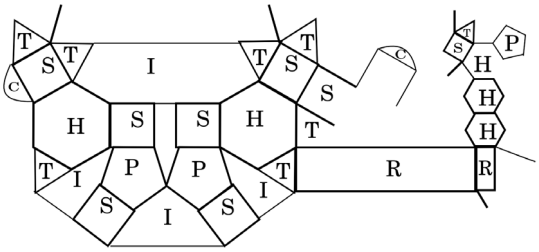

Fig. 18. The figure used to evaluate the learned concepts of polygons is illustrated. Labels of different figures given by 10 independent non-expert users are shown in the image (T = *triangle*, R = *rectangle*, S = *square*, H = *hexagon*, P = *pentagon*, I = *irregular polygon*, C = *curve*).

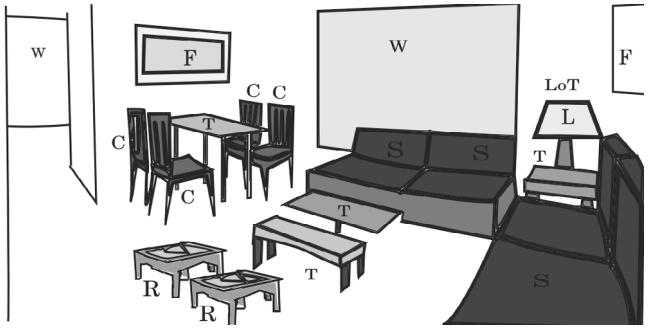

Fig. 19. The scene of a real living/dinner room used to evaluate the concepts learned about furniture is illustrated. Labels of different elements of furniture and their relations given by 10 independent non-expert users are shown in the image (*Objects* : F = *framed picture*, C = *chair*, L = *lamp*, T = *table*, W = *window*, R = *foot rest*, S = *sofa*; *Relations* : o = *on* (*e.g.* LoT = *lamp on table*)).

we presented a novel algorithm that incrementally discovers relational concepts while exploring an unknown environment. It identifies instances of potential concepts using a graph-based representation and a sub-graph discovery algorithm. Similar sub-graphs discovered are grouped together and general concept definitions are induced using an ILP algorithm. We tested the approach on three domains with encouraging results.

As future work we would like to include an intelligent exploration strategy that could be coupled with the concept learning algorithm, borrowing some ideas from intrinsically motivated reinforcement learning. We are currently not inducing recursive definitions, including functional symbols or negated literals but we plan to incorporate them in the future. Also, an extension to produce definitions of concepts from unconnected sub-graphs can be done. We would also like to test the usefulness of the learned concepts to perform simple navigation and manipulation tasks. Also, we are exploring how to learn compound actions to learn complex tasks.

Acknowledgment

This work was done under partial support of CONACYT (Ph. D. Scholarship 224491).

Appendix. Definitions of the concepts identified by the users

See Tables A.11–A.13.

Table A.11
Floors. Definitions of the concepts identified by the expert user and the majority of the non-expert users. For presentation purposes, the number of arguments of previously learned concepts used in the definition of new concepts is reduced to one.

Floors		
Concepts	Expert	Users
c1(A,B,C,D,E):-wall(A),wall(B),wall(C),wall(D),wall(E), touches(A,E),touches(B,C),touches(C,D),touches(D,E).	*room*	
c3(e1,e2):-c1(e1),wall(e2),touches(e1,e2).	*room with wall*	*room*
c4(A,B,C,D):-c1(A),c1(B),c1(C),c1(D),touches(A,D),touches(B,C),touches(C,D).	*set of rooms*	
c8(q1,q2,q3,q4):-c1(q1),c1(q2),wall(q3),wall(q4), touches(q3,q4),touches(q1,q4),touches(q1,q2).	*two rooms and wall*	
c9(s1,s2,s3,s4,s5):-c1(s1),c1(s2),c1(s3),c1(s4),wall(s5), touches(s1,s5),touches(s2,s5),touches(s2,s3),touches(s1,s4).	*set of rooms and wall*	
c10(d1,d2,d3):-c1(d1),wall(d2),wall(d3),touches(d2,d3),touches(d1,d3).	*room and two walls*	*room*
c15(b1,b2,b3):-c4(b1),c1(b2),c1(b3),touches(b2,b3),touches(b1,b3).	*two rooms and set of rooms*	*set of rooms*

Table A.12
Polygons. Definitions of the concepts identified by the expert user and the majority of the non-expert users. For presentation purposes, the number of arguments of previously learned concepts used in the definition of new concepts is reduced to one.

Polygons		
Concepts	Expert	Users
c1(A,B,C,D):-line(A),line(B),line(C),line(D), angcx90(A,B),angcx90(A,D),angcx90(B,C),angcx90(C,D).	*square*	*square*
c5(A,B,C,D,E,F): line(A),line(B),line(C),line(D),line(E),line(F), angcx120(A,B),angcx120(A,F),angcx120(B,C),angcx120(C,D),angcx120(D,E),angcx120(E,F).	*hexagon*	*hexagon*
c7(A,B,C,D,E):-line(A),line(B),line(C),line(D),line(E), angcx108(A,B),angcx108(A,E),angcx108(B,C),angcx108(C,D),angcx108(D,E).	*pentagon*	*pentagon*
c8(A,B,C):-line(A),line(B),line(C),angcx60(A,B),angcx60(A,C),angcx60(B,C).	*triangle*	*triangle*
c9(t1,t2):-c1(t1),c1(t2),angcx90(t1,t2),angcx90(t1,t2).		*two squares*
c11(y1,y2):-c8(y1),c8(y2),angcx90(y1,y2).	*two triangles*	
c18(v1,v2,v3,v4,v5):-c1(v1),line(v2),line(v3),line(v4),line(v5), angcc315(v2,v3),angcx67(v1,v3),angcx90(v1,v4),angcx67(v2,v4),angcx90(v1,v5).	*irregular polygon*	
c19(t1,t2):-line(t1),line(t2),angcx130(t1,t2).	130° *angle*	
c21(A,B):-line(A),line(B),angcx45(A,B).	45° *angle*	
c23(y1,y2):-c21(y1),line(y2),angcc315(y1,y2).	45° *angle and* 315° *angle*	
c26(o1,o2):-line(o1),line(o2),angcx67(o1,o2).	67° *angle*	
c27(s1,s2):-c26(s1),line(s2),angcx108(s1,s2).	67° *angle and* 108° *angle*	
c28(c1,c2):-c7(c1),line(c2),angcx108(c1,c2).	*pentagon and* 108° *angle*	
c29(w1,w2):-c1(w1),c8(w2),angcx90(w1,w2).	*square and triangle*	
c30(v1,v2):-c8(v1),c8(v2),angcx60(v1,v2).	*two triangles*	
c32(m1,m2):-c5(m1),c5(m2),angcx120(m1,m2).	*two hexagons*	

Table A.13
Furniture. Definitions of the concepts identified by the expert user and the majority of the non-expert users. For presentation purposes, the number of arguments of previously learned concepts used in the definition of new concepts is reduced to one.

Furniture		
Concepts	Expert	Users
c1(A,B,C,D,E):-flat_board(A),leg(B),leg(C),leg(D),leg(E),on(A,B),on(A,C),on(A,D),on(A,E).	*table*	*table*
c2(A,B,C,D):-back(A),seat(B),leg(C),leg(D),behind(A,B),on(B,C),on(B,D).	*chair of two legs*	*part of chair*
c3(A,B,C,D,E):-footrest(A),leg(B),leg(C),leg(D),leg(E),on(A,B),on(A,C),on(A,D),on(A,E).	*footrest*	*footrest*
c4(A,B,C):-c2(A),leg(B),leg(C),on(A,B),on(A,C).	*chair*	
c5(o1,o2,o3):-c1(o1),lampshade(o2),light_bulb_base(o3),on(o2,o3),on(o3,o1).	*lamp on table*	
c7(A,B,C):-seat(A),leg(B),leg(C),on(A,B),on(A,C).	*chair of two legs*	
c8(j1,j2):-c7(j1),leg(j2),on(j1,j2).	*stool*	
c10(y1,y2):-c5(y1),c2(y2),in_front_of(y1,y2).	*chair in front of table with lamp*	
c13(y1,y2):-c2(y1),c2(y2),next_to(y1,y2).	*two chairs*	
c16(l1,l2):-c2(l1),arm_support(l2),next_to(l1,l2).	*chair and arm support*	
c17(j1,j2):-c13(j1),framed_picture(j2),above(j2,j1).	*framed picture above chairs*	
c18(o1,o2,o3,o4):-lampshade(o1),light_bulb_base(o2), flat_board(o3),leg(o4),on(o1,o2),on(o2,o3),on(o3,o4).	*lamp on table*	*lamp on table*

References

[1] R. Sutton, A. Barto, Introduction to Reinforcement Learning, first ed., MIT Press, Cambridge, MA, USA, 1998.
[2] G. Bekey, Autonomous Robots: From Biological Inspiration to Implementation and Control, MIT Press, 2005.
[3] S. Wrobel, Concept formation during interactive theory revision, Mach. Learn. 14 (2) (1994) 169–191.
[4] I. Stahl, Predicate invention in inductive logic programming, in: L.D. Raedt (Ed.), Advances in Inductive Logic Programming, IOS Press, Ohmsha, Amsterdam, 1996, pp. 34–47.
[5] S. Muggleton, W. Buntine, Machine invention of first-order predicates by inverting resolution, in: Proceedings of the 5th International Conference on Machine Learning, Morgan Kaufmann, Los Altos. CA, USA, 1988, pp. 339–352.
[6] R. Wirth, Learning by failure to prove, in: D. Sleeman (Ed.), Proceedings of the 3rd European Working Session on Learning, Pitman, London, UK, 1988, pp. 237–251.
[7] J. Wogulis, P. Langley, Improving efficiency by learning intermediate concepts, in: Proceedings of the 11th International Joint Conference on Artificial Intelligence, Vol. 1, Morgan Kaufmann, San Francisco, CA, USA, 1989, pp. 657–662.
[8] P. Flach, Predicate invention in inductive data engineering, in: Proceedings of the European Conference on Machine Learning, in: Lecture Notes in Computer Science, vol. 667, Springer-Verlag, London, UK, 1993, pp. 83–94.
[9] K. Morik, S. Wrobel, J. Kietz, W. Emde, Knowledge Acquisition and Machine Learning: Theory, Methods, and Applications, Knowledge-Based Systems, Academic Press, 1993.
[10] M. Bain, S. Muggleton, Non-monotonic learning, in: Inductive Logic Programming, Academic Press, London, 1992, pp. 145–161.
[11] H. Bostrom, Predicate invention and learning from positive examples only, in: Proceedings of the Tenth European Conference on Machine Learning, in: Lecture Notes in Computer Science, vol. 1398, Springer, London, UK, 1998, pp. 226–237.
[12] J. Davis, E. Berg, D. Page, V.S. Costa, P. Peissig, M. Caldwell, Discovering latent structure in clinical databases, in: Proceedings from NIPS 2011 Workshop: From Statistical Genetics to Predictive Models in Personalized Medicine, 2011.
[13] K. Inoue, K. Furukawa, I. Kobayashi, Abducing rules with predicate invention, in: Proceedings of the 19th International Conference on Inductive Logic Programming, in: Lecture Notes in Artificial Intelligence, vol. 667, Springer-Verlag, 2009, pp. 83–94.
[14] K. Stanley, P. Domingos, Statistical predicate invention, in: Proceedings of the 24th International Conference on Machine Learning, ACM, New York, NY, USA, 2007, pp. 433–440.
[15] U. Schmid, M. Hofmann, E. Kitzelmann, Inductive programming: Example-driven construction of functional programs, Künst. Intell. 23 (2) (2009) 38–41.
[16] N. Li, D. Stracuzzi, P. Langley, Learning conceptual predicates for teleoreactive logic programs, in: Proceedings of the Late-Breaking Papers Track at the Eighteenth International Conference on Inductive Logic Programming, 2008, pp. 75–80.
[17] G. Leban, J. Zabkar, I. Bratko, An experiment in robot discovery with ilp, in: F. Zelezný, N. Lavrac (Eds.), Proceedings of the 18th International Conference on Inductive Logic Programming, in: Lecture Notes in Computer Science, vol. 5194, Springer, Berlin, Heidelberg, 2008, pp. 77–90.
[18] A. Kosmerlj, I. Bratko, J. Zabkar, Embodied concept discovery through qualitative action models, Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 19 (3) (2011) 453–475.
[19] K. Stanley, P. Domingos, Statistical predicate invention, in: Proceedings of the 24th Annual International Conference on Machine Learning, 2007, pp. 433–440.
[20] G. Leban, J. Zabkar, I. Bratko, An experiment in robot discovery with ILP, in: F. Zelezný, N. Lavrac (Eds.), Inductive Logic Programming, in: Lecture Notes in Computer Science, vol. 5194, Springer, Berlin, Heidelberg, 2008, pp. 77–90.
[21] L. Holder, D. Cook, S. Djoko, Substructure discovery in the subdue system, in: Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, AAAI Press, 1994, pp. 169–180.

[22] B. Chien, C. Hu, M. Ju, Learning fuzzy concept hierarchy and measurement with node labeling, Inf. Syst. Front. 11 (2009) 551–559.
[23] J. Rosca, Hierarchical learning with procedural abstraction mechanisms (Ph.D. thesis), Department of Computer Science, The College of Arts and Sciences, University of Rochester, Rochester, NY 14627, USA, 1997.
[24] R. Rivest, R. Sloan, A formal model of hierarchical concept learning, Inform. Comput. 114 (1994) 88–114.
[25] B. Zupan, M. Bohanec, I. Bratko, J. Demšar, Learning by discovering concept hierarchies, Artificial Intelligence 109 (1999) 211–242.
[26] J. Tani, S. Nolfi, Learning to perceive the world as articulated: An approach for hierarchical learning in sensory-motor systems, Neural Netw. 12 (1999) 1131–1141.
[27] L. Fu, B. Buchanan, Learning intermediate concepts in constructing a hierarchical knowledge base, in: Proceedings of the 9th International Joint Conference on Artificial intelligence, Vol. 1, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1985, pp. 659–666.
[28] S. Muggleton, D. Lin, N. Pahlavi, A. Tamaddoni-Nezhad, Meta-interpretive learning: application to grammatical inference, Mach. Learn. 94 (1) (2014) 25–49.
[29] S. Muggleton, D. Lin, A. Tamaddoni-Nezhad, Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited, Mach. Learn. 100 (1) (2015) 49–73.
[30] A. Cropper, S. Muggleton, Learning efficient logical robot strategies involving composable objects, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence, 2015, pp. 3423–3429.
[31] S. Muggleton, Inverse entailment and progol, New Gener. Comput. 13 (3–4) (1995) 245–286.
[32] J. Sowa, Conceptual Graphs, Elsevier B. V., 2008, pp. 213–237. (Chapter 5).
[33] D. Cook, L. Holder, Substructure discovery using minimum description length and background knowledge, J. Artificial Intelligence Res. 1 (1) (1994) 231–255.
[34] S. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third ACM Symposium on Theory of Computing, ACM, New York, NY, USA, 1971, pp. 151–158.
[35] L. Holder, D. Cook, Subdue, graph based knowledge discovery, 1994. URL http://ailab.wsu.edu/subdue/.
[36] A. Tenorio, Automatic discovery of concepts, 2015. URL https://sites.google.com/site/automaticdiscoveryconcepts/.
[37] R.V. Gerkey, B.A. Howard, The player/stage project: Tools for multi-robot and distributed sensor systems, in: Proceedings of the 11th International Conference on Advanced Robotics, 2003, pp. 317–326.
[38] V. Nguyen, S. Gächter, A. Martinelli, N. Tomatis, R. Siegwart, A comparison of line extraction algorithms using 2D range data for indoor mobile robotics, Auton. Robots 23 (2) (2007) 97–111.

Ana C. Tenorio-González received a B.E. degree in Computational Systems from the Instituto Tecnológico Superior de Xalapa, Mexico in 2008, and a M.Sc. degree in Computer Science from the Instituto Nacional de Astrofísica, Optica y Electrónica (INAOE), Mexico in 2010. She is interested in the areas of machine learning, pattern recognition and robotics. She has worked on reinforcement learning with reward shaping and intrinsic motivation; and concept learning, both topics applied to robotics. Currently, she is Ph.D. student at INAOE.

Eduardo F. Morales received his B.Sc. degree in Physics Engineering from Universidad Autonoma Metropolitana, Mexico City, his M.Sc. degree in Information Technology: Knowledge-based Systems from the University of Edinburgh, and his Ph.D. degree in Computer Science from the Turing Institute—University of Strathclyde, Scotland. He has been responsible for more than 18 research projects sponsored by different funding agencies and has more than 100 articles in journals, book's chapters and conference proceedings. He is currently a research scientist of the Instituto Nacional de Astrofsica, Optica y Electrónica (INAOE) in Mexico where he conducts research in Machine Learning and Robotics.