Carnegie
Mellon

Ecobweb:

Preliminary User's Manual

Yoram Reich
Present Address:
Department of Solid Mechanics Materials and Structures
Faculty of Engineering
Tel Aviv University
Ramat Aviv 69978
Israel
email: yoram@eng.tau.ac.il
http://or.eng.tau.ac.il:7777/

Original report date: June 1, 1991

Department of Civil Engineering
Carnegie Institute of Technology
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890
(412) 268-2940

Contents

List of Figures

List of Tables

1 Introduction

The following lines warn you about the status of the program.

- Ecobweb was developed as a research project. While it was tested on many domains, its stage may be described as being a prototype program.
- Ecobweb is not fully documented. This manual gives a partial account of the main functions only, but those that will get you most of the functionality.
- No provision for extensive checking of input or selection of parameters is coded.
- (there are probably inconsistencies in defining functions (read-example-set, read-tree))

Since Ecobweb is still under development, I will appreciate getting bug reports, suggestion for useful features, and comments. Please send them to yoramcmu.edu
The following Copyright notice appears in the all the program files. Do not remove it:

```
;;;-------------------------------------------------------------------------
;;;   Ecobweb 0.99 released 11/20/92   contact: yoram@cmu.edu
;;;   by Yoram Reich
;;;   Engineering Design Research Center, Carnegie Mellon University,
;;;   5000 Forbes Ave., Pittsburgh, PA 15213, USA
;;;
;;;   Copyright (C) 1992 Yoram Reich
;;;   All rights reserved.  This software comes with NO WARRANTY of any kind.
;;;   This software may not be distributed to any other party without
;;;   explicit permission from Yoram Reich. This software is distributed
;;;   for research purposes only.
;;;-------------------------------------------------------------------------
```

2 About ECOBWEB

ECOBWEB is a system that learns and uses knowledge for synthesis (Reich and Fenves, 1992). It is the core part of BRIDGER, a system developed to explore the utility of using machine learning techniques for assisting in the design of cable-stayed btidges (Reich, 1991a). ECOBWEB is re-implementation with extensions of the concept formation learning program COBWEB (Fisher, 1987) to synthesis tasks. Therefore, many properties of ECOBWEB discussed here are inherited from its predecessor. Since the purpose of this paper is not the evaluation of ECOBWEB, we do not emphasize the distinctions between the two learning systems. Such evaluations are described elsewhere (Reich, 1991a; Reich and Fenves, 1991; Reich and Fenves, 1992). Rather, this section reviews the assumptions underlying ECOBWEB and illustrates its operation using the chair domain which is described in Appendix A.

2.1 Assumptions

ECOBWEB makes the following assumptions about design that are believed to be appropriate for the preliminary design of a large class of design problems.

ASSUMPTION 1 (**Artifact representation**): Artifacts and their specifications are described by (finite) lists of property-value pairs.

This assumption restricts the scope of designs to those with fixed topologies. Therefore, the design of artifacts that are described via graphs, such as layouts, cannot be supported under this assumption. If the restriction on the finite number of properties is removed, the representation becomes general, but of course, impossible to implement computationally.

ASSUMPTION 2 (**Structure of design knowledge**): Design knowledge is represented in a hierarchical classification tree.

This assumption does not impose any restrictions on the potential application. The crucial issue is how the knowledge structure is being used.

ASSUMPTION 3 (**Quality of design knowledge**): The quality of design knowledge is recursively determined by a syntactic function called category utility.

To elaborate the assumption, ECOBWEB evaluates a classification of a set of designs into mutually-exclusive classes $C_1, C_2, \ldots, C_n$ by a statistical function called *category utility (CU)*:

$$CU = \frac{\sum_{k=1}^{n} P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2}{n} \quad (1)$$

where C_k is a class, $A_i = V_{ij}$ is a property-value pair, $P(x)$ is the probability of x, and n is the number of classes. The first term in the numerator measures the expected number of property-value pairs that can be guessed correctly by using the classification. The second term measures the same quantity *without* using the classes. Thus, the category utility measures the expected *increase* of property-value pairs that can be guessed *above* the guess based on frequency alone. The measurement is normalized with respect to the number of classes. The higher is the value of CU, the better the quality of the knowledge is.

The knowledge quality assumption approximates the more desired quality measure stating that a classification is 'good' if the description of a design can be guessed with high accuracy, given that it belongs to a specific class, or even the better measure, that a classification is 'good' if it results in good design performance. Although the quality assumption compromises quality, it supports the use of an efficient algorithm for building the knowledge[1].

> ASSUMPTION 4 (**Design process**): Design is a *direct* mapping from the specification to the artifact description.

Defining design as a mapping is very general unless the nature of the mapping is restricted to be *direct* as it is in the above assumption. This assumption almost excludes the use of explicit knowledge in design. On the other hand, information can be compiled into implicit knowledge embedded in the mapping.

> ASSUMPTION 5 (**Nature of design knowledge**): The structure of design knowledge is static. It can only be altered incrementally if new design knowledge warrants it.

This assumption states that knowledge is believed to be correct, unless additional information becomes available. The nature of the additional information is restricted to the description of new designs and to procedures that evaluate the knowledge quality. Another consequence of this assumptions is that knowledge is built incrementally and continuously since it will always be incomplete.

Assumption 5 determines that a system that supports design must be able to learn and modify its content incrementally. Assumption 2 establish the structure of the knowledge that needs to be generated and Assumption 3 determines how this knowledge should be evaluated. Assumption 1 restricts the scope of artifact discussed, thereby allowing available learning techniques to be used for the knowledge generation. Of course, all these assumptions lead to the selection of COBWEB (Fisher, 1987) as a potential tool. Since COBWEB has several limitations in the context of design (Reich, 1991a; Reich and Fenves, 1991), a new system was developed, called ECOBWEB, that will be able to support all the assumptions in design domains.

The next two subsections describe the realization of these assumptions in ECOBWEB learning and synthesis processes.

2.2 ECOBWEB's learning algorithm

ECOBWEB learns from a sequence of design examples. Examples need not be classified as feasible, optimal, or by any other classification scheme. However, any *a priori* classification can be assigned to an example and treated as any other property. When a new design is introduced, ECOBWEB tries to accommodate it into the existing classification hierarchy starting at the root. The system performs one of the following operators (see (Fisher, 1987) for a detailed description of these operators):

(1) expanding the root, if it does not have any sub-classes, by creating a new class and attaching the root and the new design as its sub-classes;
(2) adding the new design as a new sub-class of the root;

[1] In this paper, we are using the terms design knowledge and its quality rather loosely; elsewhere, we try to address these issues in more detail while pointing at some of the difficulties in being precise about them (Reich, 1995).

(3) adding the new design to one of the sub-classes of the root;
(4) merging the two best sub-classes and putting the new design into the merged sub-class; or
(5) splitting the best sub-class and again considering all the alternatives.

If the design has been assimilated into an existing sub-class, the process recurses with this class as the top of a new hierarchy. Ecobweb uses the category utility function (CU) to evaluate the results of the operator applications and selects the operator that results in the highest CU score.

Figure 1 shows the hierarchy generated from the eight chairs (see Part I of the study (Reich, 1991b) for a detail description) by Ecobweb. The classes are described with all their properties. The properties that are shown in bold font are the characteristic properties. Intuitively, characteristic property values of a class are those property values that are very common in the class and rarely appear in the other classes of the same level. Translating this intuition into probability terminology, characteristics are property values that satisfy $P(A_i = V_{ij}|C_k) \geq \mathit{threshold}$ and $P(C_k|A_i = V_{ij}) \geq \mathit{threshold}$, where $\mathit{threshold}$ is a pre-determined fixed value that potentially, can be learned for each domain. The figure also shows the name of each class and the value of $P(A_i = V_{ij}|C_k)$, denoted by P, for each property value of an abstract concept.

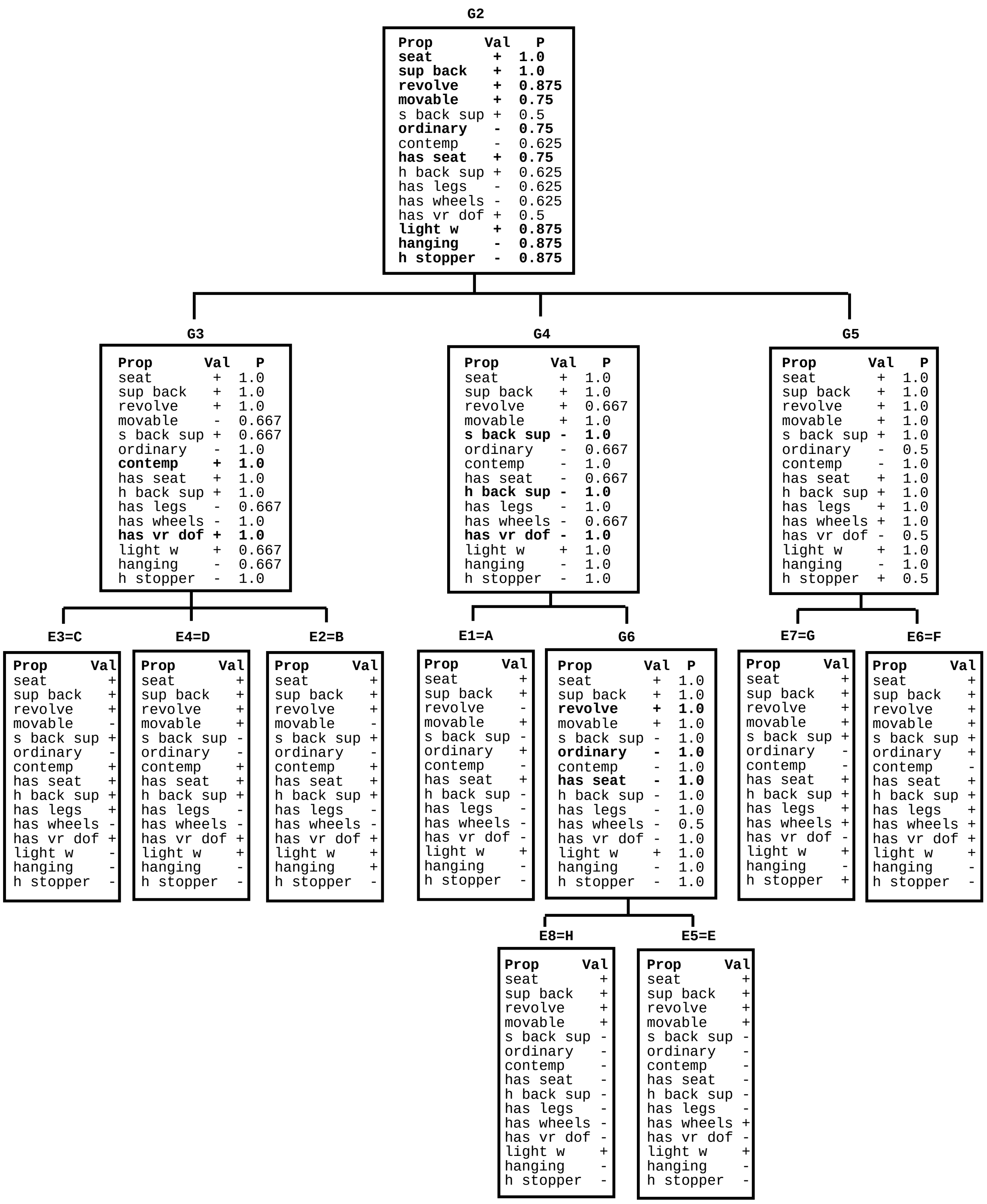
2.3 Ecobweb's synthesis process

Ecobweb has several synthesis methods which can be described along two dimensions: the refinement dimension which can be *extensional* or *intentional*; and the generation dimension which can be *case-based* or *prototype-based*. Figure 2 illustrates these dimensions. In the extensional approach, refinement classifies a new specification with a new subclass starting from the top class (class 1 in Figure 2) until the process terminates at class 3. In the intentional approach, while classifying the new specification, characteristic property-values of the classes traversed (classes 1, 2, and 3 in Figure 2) are assigned to the new design. In the generation stage, the case-based approach views a class as representing the extension of all its leaves. Therefore, leaves 4, 5, and 6 are selected as candidates in conjunction with the extensional refinement approach, or are used to complete the missing properties in conjunction with the intentional refinement approach. The prototype-based generation approach takes the current class (class 3 in the example) and uses its property-value pairs to create candidate designs in conjunction with the extensional refinement approach, or to complete several descriptions in conjunction with the intentional refinement approach.

Three design scenarios from the chair domain illustrate the synthesis techniques. The design scenarios assume that the current state of knowledge is as appears in Figure 1.

Example 1: The first design scenario deals with designing a chair that is *movable*, *contemporary* and *stably support back*. Even though, the set of examples that satisfy this specification is empty, Ecobweb still proceeds to synthesize a candidate. In the *case-based/extensional* method, Ecobweb starts by selecting G5 for further refinement. It is interesting to note that the selection between classes G5 and G3 is almost arbitrary since both are almost equally good for the first refinement step. The final candidate is a random choice between chairs F and G, which both satisfy two of the three specification properties. In the *case-based/intentional* method, the characteristic property values of G2 is used to refine the specification (revolve, seat, support back) and the design (light-weight, not hanging, no stopper). Again, G5 is selected for further refining the design and the final result is the addition of the properties of F to the current partial design description. The two *prototype* approaches yield the same behavior since the process does not end at an intermediate class; therefore there is no "true" prototype that can generate several alternatives.

Example 2: The second design scenario deals with designing a chair that *revolves* and *is movable*. In the *case-based/extensional* method, Ecobweb stops at class G2 since both the specification properties are matches by characteristic values. Therefore, all the eight chairs are candidate designs. In the *case-based/intentional* method, the characteristic property values of G2 are used to refine the specification



G2

Prop	Val	P
seat	**+**	**1.0**
sup back	**+**	**1.0**
revolve	**+**	**0.875**
movable	**+**	**0.75**
s back sup	+	0.5
ordinary	**-**	**0.75**
contemp	-	0.625
has seat	**+**	**0.75**
h back sup	+	0.625
has legs	-	0.625
has wheels	-	0.625
has vr dof	+	0.5
light w	**+**	**0.875**
hanging	**-**	**0.875**
h stopper	**-**	**0.875**

G3

Prop	Val	P
seat	+	1.0
sup back	+	1.0
revolve	+	1.0
movable	-	0.667
s back sup	+	0.667
ordinary	-	1.0
contemp	**+**	**1.0**
has seat	+	1.0
h back sup	+	1.0
has legs	-	0.667
has wheels	-	1.0
has vr dof	**+**	**1.0**
light w	+	0.667
hanging	-	0.667
h stopper	-	1.0

G4

Prop	Val	P
seat	+	1.0
sup back	+	1.0
revolve	+	0.667
movable	+	1.0
s back sup	**-**	**1.0**
ordinary	-	0.667
contemp	-	1.0
has seat	-	0.667
h back sup	**-**	**1.0**
has legs	-	1.0
has wheels	-	0.667
has vr dof	**-**	**1.0**
light w	+	1.0
hanging	-	1.0
h stopper	-	1.0

G5

Prop	Val	P
seat	+	1.0
sup back	+	1.0
revolve	+	1.0
movable	+	1.0
s back sup	+	1.0
ordinary	-	0.5
contemp	-	1.0
has seat	+	1.0
h back sup	+	1.0
has legs	+	1.0
has wheels	+	1.0
has vr dof	-	0.5
light w	+	1.0
hanging	-	1.0
h stopper	+	0.5

E3=C

Prop	Val
seat	+
sup back	+
revolve	+
movable	-
s back sup	+
ordinary	-
contemp	+
has seat	+
h back sup	+
has legs	+
has wheels	-
has vr dof	+
light w	-
hanging	-
h stopper	-

E4=D

Prop	Val
seat	+
sup back	+
revolve	+
movable	+
s back sup	-
ordinary	-
contemp	+
has seat	+
h back sup	+
has legs	-
has wheels	-
has vr dof	+
light w	+
hanging	-
h stopper	-

E2=B

Prop	Val
seat	+
sup back	+
revolve	+
movable	-
s back sup	+
ordinary	-
contemp	+
has seat	+
h back sup	+
has legs	-
has wheels	-
has vr dof	+
light w	+
hanging	+
h stopper	-

E1=A

Prop	Val
seat	+
sup back	+
revolve	-
movable	+
s back sup	-
ordinary	+
contemp	-
has seat	+
h back sup	-
has legs	-
has wheels	-
has vr dof	-
light w	+
hanging	-
h stopper	-

G6

Prop	Val	P
seat	+	1.0
sup back	+	1.0
revolve	**+**	**1.0**
movable	+	1.0
s back sup	-	1.0
ordinary	**-**	**1.0**
contemp	-	1.0
has seat	**-**	**1.0**
h back sup	-	1.0
has legs	-	1.0
has wheels	-	0.5
has vr dof	-	1.0
light w	+	1.0
hanging	-	1.0
h stopper	-	1.0

E7=G

Prop	Val
seat	+
sup back	+
revolve	+
movable	+
s back sup	+
ordinary	-
contemp	-
has seat	+
h back sup	+
has legs	+
has wheels	+
has vr dof	-
light w	+
hanging	-
h stopper	+

E6=F

Prop	Val
seat	+
sup back	+
revolve	+
movable	+
s back sup	+
ordinary	+
contemp	-
has seat	+
h back sup	+
has legs	+
has wheels	+
has vr dof	+
light w	+
hanging	-
h stopper	-

E8=H

Prop	Val
seat	+
sup back	+
revolve	+
movable	+
s back sup	-
ordinary	-
contemp	-
has seat	-
h back sup	-
has legs	-
has wheels	-
has vr dof	-
light w	+
hanging	-
h stopper	-

E5=E

Prop	Val
seat	+
sup back	+
revolve	+
movable	+
s back sup	-
ordinary	-
contemp	-
has seat	-
h back sup	-
has legs	-
has wheels	+
has vr dof	-
light w	+
hanging	-
h stopper	-

Figure 1: A classification hierarchy of the chair domain generated by Ecobweb. The property names are obvious abbreviations of those appearing in Tables 1 and 2 of Appendix A.

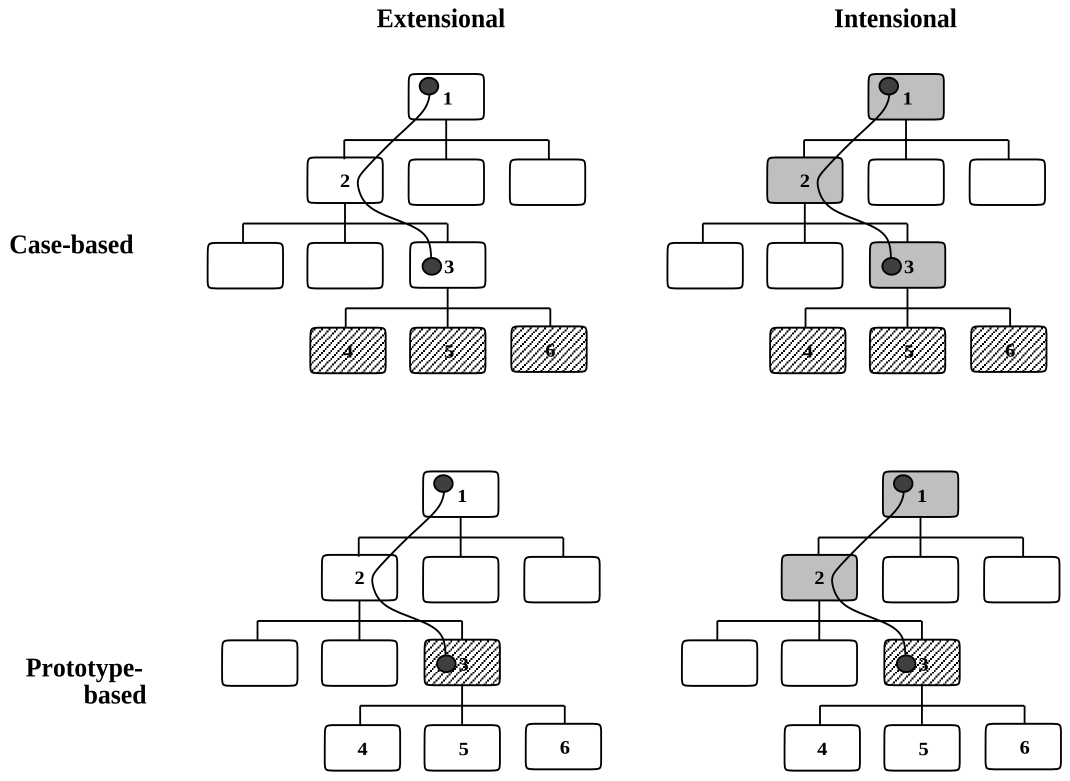


Figure 2: Synthesis methods

(ordinary, seat, support back) and the design (has seat, light-weight, not hanging, no stopper). The eight chairs that are again candidates are only used to *complete* the partial description accumulated thus far from the characteristic values. In the *prototype/extensional* method, ECOBWEB generates 14 candidates from the various com binations of property values represented in class G2. The *prototype/intentional* approach yields 14 candidates from variations on property values that are not characteristics.

EXAMPLE 3: The third problem is to design a *contemporary* chair. The first refinement step chooses class G3. In the *case-based/extensional* method, ECOBWEB selects chairs B, C, D as candidates. In the *case-based/intentional* method, ECOBWEB generates 6 candidates from class G3. As in example 1, The two *prototype* approaches yield the same behavior since the process does not end at an intermediate class.

There are two observations from the three examples and other examples not described here. First, ECOBWEB always generates alternatives that not always satisfy all the requirements and it generates alternatives that *did not exist* in the original set of chairs. Second, in deep hierarchies generated by many examples, it is observed that the path traversed by the guidance of the category utility function can be interpreted as a progressive matching of the specifications or even as a design derivation[2]. this behavior is desirable, even though the coherence of the knowledge structure generated is not conceptualized as a criterion for the success of the learning approach.

2.4 performance

ECOBWEB was tested in many domains (Reich, 1991a) and demonstrated performance comparable and often better than other learning programs in classification domains. but more important, it demonstrated good performance in design domains (Reich, 1991a). to illustrate the performance, we review some results of ECOBWEB's evaluation in the domain of cable-stayed bridge design (Reich, 1991a).

Table 1: scaling statistics of candidates

knowledge	scaling
k_1	3.07
k_2	2.15
k_3	2.09
k_4	1.32

table 1 illustrates the performance of ECOBWEB by providing the values of one performance measure generated from statistics of 48 test problems. the measure, called *scaling*, calculates how close was the span of the synthesized bridge to the required span. the measure is provided for four knowledge hierarchies, k_1, k_2, k_3, and k_4, generated by learning. hierarchy k_1 was generated from the original 96 bridge examples. hierarchy k_2 was generated from the 96 examples after their analysis and redesign; therefore it contains higher quality examples. hierarchies k_3 and k_4 were generated from 144 and 192 good quality examples, respectively. since k_1 was built from lower-quality examples it does not participate in the statistical analyses performed. two models were hypothesized: (1) scaling linearly depends on the number of examples in a hierarchy; and (2) the logarithm of scaling depends on the logarithm of the number of examples. the latter reflect the power low of practice (Newell and Rosenbloom, 1981).

[2]it is important to acknowledge that the sequence of design description property-value assignments does not approximate in any way the explicit intent and domain knowledge on the order in which design derivations are to proceed. such concerns may be supported when domain knowledge is incorporated in the learning or synthesis processes.

a general linear model (glm) procedure with a MANOVA analysis was performed to assess the differences between the scaling values observed according to the two models. in both models, the scaling values satisfy: $k_2, k_3 >_{0.01} k_4$; where the $>_{0.01}$ indicates that k_2 and k_3 are greater than k_4 with statistical significance at the $p < 0.01$ level and that the difference between k_2 and k_3 was not statistically significant. note that the second model was better than the first, although the statistical significance of this statement was not assessed. therefore, *the more knowledge* ECOBWEB *has, the more relevant are the retrieved candidates.* the improvement is not a smooth function, but occurs in steps. this performance evaluation shows that ECOBWEB captures knowledge and gradually uses it more effectively in design.

3 Program structure

3.1 Files

undocumented

3.2 Terminology

ctree : classification tree

example-set : a set of examples

example : an example data structure

depth : depth of the node measured from the root

property-list : a list of properties

step : number of examples learned before the next prediction

cardinality : number of examples in the set

learning-set : examples used to create the classification tree

partial-examples : a set of examples to be predicted, assumed to be missing the predicted property

4 Getting started

4.1 Organizing your directory

The distribution file is an encoded compressed "tape." To retrieve the files:

1. Remove mail header (until begin 644) and store in a file, say "foo".
2. uudecode foo
3. uncompress ecobweb.tar.Z
4. tar xvf ecobweb.tar
5. Then look for "README" file

The following files are important:

definitions.l contains definitions of the domain

Table 2: Parameters most commonly used

name	default	for more info
cobweb-type	regular	5.1
random-state	machine generated	
design-description	nil	4.3
property-list	nil	4.3
specifications	nil	4.3
misc-descriptors	nil	4.3
continuous-classes	8	
prediction-method	leaf	5.2
property-types	global hash table	
expected-interval	global hash table	
P.v-cl	0.75	
P.v-cl2	0.85	
P.cl-v	0.75	
prediction-utility	nil	
continuous-ranges	static, adaptive, or dynamic	

Table 3: Parameters that should be added

name	default	for more info
print-utility	nil	
print-tree	nil	
print-operators	nil	

test.l contains the parameters and specific functions to be performed

run.l load and compile (if necessary) the program

examples contains the examples in the required format

4.2 Global parameters

These parameters control various aspects of ECOBWEB. Many of them were used in the development and are not required too much attention. it is important to have them all, otherwise lisp will complain.

Use the existing files as templates and modify their values.

The following usually stay fixed:

4.3 Preparing your data

Several files need to be prepared before running ECOBWEB

4.3.1 Defining the domain

All properties are set to be nominal by default. They should be declared if they are different (continuous, hierarchical (not supported yet)).

For continuous properties, approximate expected range of each property should be given; ECOBWEB is not too sensitive to the choices of ranges.

4.3.2 Describe your data

Prepare a data file containing your examples.

4.3.3 Choose your environment and functions

Set global parameters.

Choose the desired function to run.

4.4 Running ECOBWEB

ECOBWEB should work in (almost) any common-lisp environment. Experience has shown that Allegro Common-Lisp is preferable whenever possible.

Before you start lisp is is useful to increase your available memory by typing:

```
unlimit datasize
```

at your Unix propmt. Make sure you don't affect other users of the machine. Type cl (or other such as lisp) to start Lisp. At the lisp interpreter type the following:

```
(load "run.l")
(load "test.l")
```

This will load (and compile, if necessary) ECOBWEB's files based on the particular machine and lisp and run your specific functions (look inside make.l to see whether your platform is supported).

5 Basic functions

This section describes pre-defined functions that can be used to learn, print, and test the results. Each function is described in turn with the global parameters that influence its execution.

5.1 Learning

```
(top-cobweb example ctree)
```

cobweb-type

regular — This will cause the regular ECOBWEB program to run. The operators that apply are the four original operators of COBWEB.

full — This will cause four other operators to be considered.

double This will run the original ECOBWEB with an example set that was doubled

classit This will run CLASSIT on the data. Make sure all the properties are continuous.

```
(top-cobweb-on-example-set example-set ctree)
```

5.2 Prediction

```
(predict-by-method partial-examples  tree)
```

The *prediction-scheme* parameter governs the type of prediction performed.

5.3 IO

5.3.1 Input

```
(read-example-set)

(read-ctree file)
```

5.3.2 Output

```
(prety-print-ctree ctree)

(prety-print-ctree-limited ctree depth)

(prety-print-ctree-limited-normatives ctree depth)

(print-example example)

(compact-print-example example property-list)

(print-example-set example-set)

(print-ctree-into-file ctree file)

(print-example-set-into-file example-set file)
```

5.3.3 Figures

5.4 Experiments

5.4.1 Background

Needs to be written! (from yoram90civil)

5.4.2 Coverage

```
(experiment cardinality step learning-set test-data print-depth)

(statistical-experiment step print-depth trials data-set
                                     &rest design-property-list)
```

5.4.3 Performance

```
(detail-experiment cardinality step learning-set print-depth
                       property-list &rest design-property-list)

(detail-statistical-experiment step print-depth trials data-set
                       property-list &rest design-property-list)

(detail-experiment-cv-k test-set training-set
                       property-list &rest design-property-list)

(detail-statistical-experiment-cv-k k trials data-set
                       property-list &rest design-property-list)
```

6 Advanced functions

experimentation

several trees

constructive induction

artificial domains

acquisition of examples

tree correction

```
(find-nodes-violating-characteristics property-set tree)

(tree-quality-correction-properties property-set tree)
```

7 Auxiliary functions

```
(divide-random-k k set)

(delete-partial-examples-from-set example-set property-list)

(query-example-set example-set query)

(choose-randomly cardinality number)

(choose-randomly-from-set set number)

(clean-descriptors-from-set example-set property-list)
```

8 Sample files

definitions.l

```
;-----------------------------------------------------------
(defparameter true t)
(defparameter *print-gensym* nil)
(defparameter *print-level* 5)
(defparameter *print-length* 30)
(defparameter *min-utility* -10)
(defparameter *start-run-time* (get-internal-run-time))
(defparameter *specifications* '(river location erected
                                purpose length lanes clear-g))
(defparameter *design-description*
        '(t-or-d material span rel-l type))
(defparameter *relevant-property-list* (append  *specifications*
                                              *design-description*))
(defparameter *misc-descriptors* '(name cost no-spans))
(defvar *domain*)
; will hold the properties values instead of going to space structures
(defvar *properties-values* (make-hash-table :test #'eq))
(defvar *debug-run* nil)
(defvar *prediction-method* 'leaf)
(defvar *P.v-cl* 0.75)
(defvar *P.cl-v* 0.75)
(defvar *property-types* (make-hash-table :test #'eql))
(dolist (property *relevant-property-list*)
     (setf (gethash property *property-types*) 'nominal))
(defvar *expected-interval* (make-hash-table :test #'eql))
(defvar *continuous-classes* 8)
```

A The domain of chairs

Figure 3 depicts eight chairs which are used to explain the concepts and ideas discussed in this study. Each chairs in the figure is denoted with a letter. The chairs provide some *functionality* that is summarized in Table 4. Each row describes a different function of a chair. The + in the table denotes that a chair provides the corresponding function, and a − denotes its lack thereof. Additional functions that chairs may have but that are not mentioned include *comfort*, *access to ceiling*, *resistance to fire*, etc. In addition to providing functions, each chair has properties that can be observed and therefore describe the *attributes* or the *structure* of the artifact; some of these are summarized in Table 5. Additional observable properties that chairs may have include *color*, *material texture*, *type of upholstery*, *dimensions*, etc.

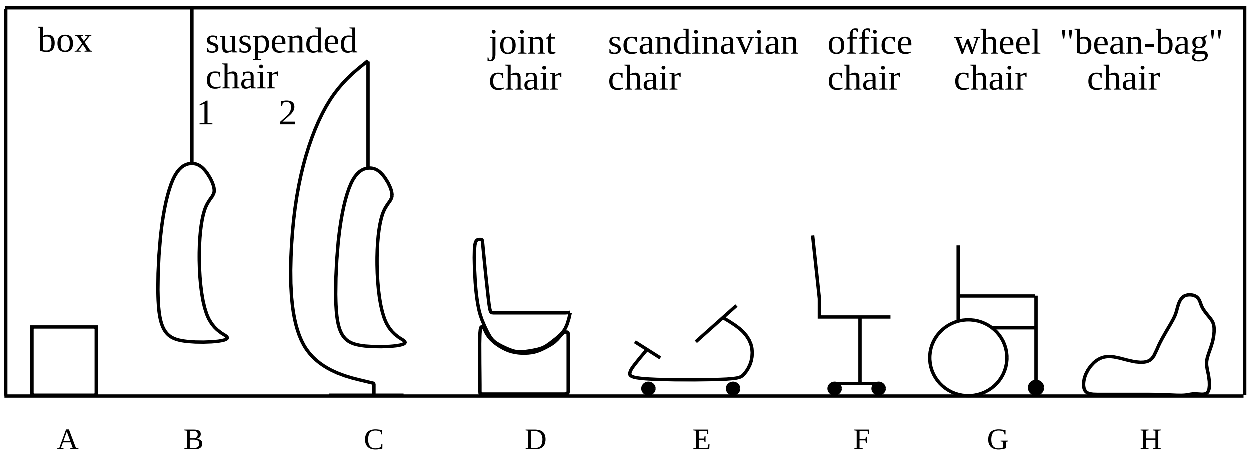


Figure 3: Chairs

Table 4: Functional properties of chairs

		chair							
	function	A	B	C	D	E	F	G	H
1	seating	+	+	+	+	+	+	+	+
2	back support	+	+	+	+	+	+	+	+
3	revolving	—	+	+	+	+	+	+	+
4	movable	+	—	—	+	+	+	+	+
5	stably support back	—	+	+	—	—	+	+	—
6	ordinary design	+	—	—	—	—	+	—	—
7	contemporary design	—	+	+	+	—	—	—	—

Naturally, there are functions that are directly derived from the structure of a chair. For example, a chair that *has wheels* is *movable* or a chair that has a *vertical rotational degree of freedom (dof)* is *revolving*. Note that this structure-function relation may be an approximation; for example, chair C with a rotational dof does not allow for 360° rotation. Other functions may be more complex and could no be inferred from one observable property. To illustrate, a chair can provide *back support* although it does not have a back. For instance, chair A provides back support due to its location near a wall. Its function is context dependent. Also, chair E provides back support due to its complex structure although is does not have a *physical* back support. Some functions may qualify other functions. For example, the function *stable back support* qualifies

Table 5: Observable properties of chairs

		chair							
	structure	A	B	C	D	E	F	G	H
1	has seat*	+	+	+	+	—	+	+	—
2	has back support	—	+	+	+	—	+	+	—
3	has legs	—	—	+	—	—	+	+	—
4	has wheels	—	—	—	—	+	+	+	—
5	has vertical rotational dof	—	+	+	+	—	+	—	—
6	has light weight	+	+	—	+	+	+	+	+
7	is hanging	—	+	+	—	—	—	—	—
8	has stopper	—	—	—	—	—	—	+	—

* A seat is a horizontal stiff object.

the function *back support*. This function is quite complex to assess. Chairs B and C are stable due to static considerations, whereas chairs D and H are not stable; chairs A, F, and G are structurally stable; and chair E does not even have a physical support.

All the previous examples have concentrated on inferring potential functionality from artifact structure. This is useful in *analysis*. We are mainly interested in *synthesis* which is the generation of artifact structure that will satisfy a desired function. For example, the specification of a chair that will be *movable* and *stably support back* leads to two potential designs F and G. These designs can be generated in two ways. The first way starts with $\{A, D, E, F, G, H\}$ as the *movable* designs and refines them with the *stably support back* property. The second way starts with $\{B, C, F, G\}$ as the *stably support back* designs and refines them with the *movable* property. The most concise description of the solution is chairs that have *physical back support* and *wheels*. Of course, this description can lead to chairs not depicted in Figure 3. The design process was made easier by the use of the eight representative chairs as mediating between the specification and the design description. In the absence of these chairs, the process is much more difficult.

As an another example, assume that in addition to the previous specification, it is also required that the chair will have a *contemporary design*. There is no chair that satisfies these three functions. A *redesign* process can be invoked by taking the current candidate designs $\{F, G\}$ and retracting either the *movable* or the *stably support back* specification properties and then trying the new specification. Since we are working with extensional description of the candidate designs, there will still not be any candidate that satisfies all the three specification properties. Nevertheless, we will have three sets of *nearly* good candidates: (1) *stably support back* and *contemporary* $\{B, C\}$, (2) *movable* and *contemporary* $\{D\}$, and (3) *movable* and *stably support back* $\{F, G\}$. If the set of designs was not confined to the eight chairs, the second group could be made *stably support back* by adding a stopper. In contrast, the first group cannot be easily made movable: B is attached to the ceiling and C would have to receive wheels and a stopper.

References

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(7):139–172.

Newell, A. and Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the power law of practice. In Anderson, J. R., editor, *Cognitive Skills and Their Acquisition*. Erlbaum Associates, Hillsdale, N.J.

Reich, Y. (1991a). *Building and Improving Design Systems: A Machine Learning Approach*. PhD thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA. (Available as Technical Report EDRC 02-16-91).

Reich, Y. (1991b). Design theory and practice I: A critical review of General Design Theory. Technical Report EDRC 12-45-91, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

Reich, Y. (1995). Measuring the value of knowledge. *International Journal of Human-Computer Studies*, 42(1):3–30.

Reich, Y. and Fenves, S. J. (1991). The formation and use of abstract concepts in design. In Fisher, D. H. J., Pazzani, M. J., and Langley, P., editors, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, pages 323–353, Los Altos, CA. Morgan Kaufmann.

Reich, Y. and Fenves, S. J. (1992). Inductive learning of synthesis knowledge. *International Journal of Expert Systems: Research and Applications*, 5(4):275–297.