SRI International

Learning Control Parameters of a Vision Process Using Contextual Information

April 6, 1994

Technical Note No. 542

By: Stephane Houzelle†, International Fellow
Thomas M. Strat, Senior Computer Scientist
Pascal Fua, Computer Scientist
Martin A. Fischler, Program Director
Artificial Intelligence Center
Computing and Engineering Sciences Division

†Currently at INRIA, Project Pastis, BP 93, F-06902 Sophia, Antipolis Cedex, FRANCE

The research reported here was funded in part by the Advanced Research Projects Agency and the U.S. Army Topographic Engineering Center under contract DACA 76-92-C-0034.

333 Ravenswood Avenue • Menlo Park, CA 94025-3493 • (415) 326-6200 • FAX: (415) 326-5512 • Telex: 334486

Abstract

Two of the problems that the user of an image understanding system must continuously face are the choice of an appropriate algorithm and the setting of its associated parameters. These requirements mean that the user must have a fairly high degree of expertise with the algorithms to accomplish a given task effectively. If, on the other hand, the system itself is able to learn how to select among its algorithms and to set their parameters through its experience with similar tasks, it should be possible to reduce the need for operator expertise while improving efficiency at the same time.

This paper presents a method to accomplish this goal. Contextual information computed from the task and the input data is used to search for similar situations, and determine whether or not an algorithm is applicable, and which parameters are suitable for it. Different approaches have been investigated as the basis for finding similar situations. The first one uses a measure of similarity between context element values. The second one uses a categorization method based on conceptual clustering. The main problem is the need to deal with both numerical and categorical variables.

To demonstrate the efficiency of our approach, we describe experiments involving the use of a snake algorithm to perform the task of curvilinear feature extraction. Our implementation allows the various parameters of this technique to be context-specific. We show in this setting how our system makes the use of a vision process easier by reducing the needed user expertise and improving efficiency in obtaining the desired results.

Keywords: Image Understanding, Contextual Information, Learning by Observation, Learning from Examples, Conceptual Clustering, Parameter Learning, Description Learning, Active Contour, Snake

Contents

1 Introduction 5

1.1 A Cartographic Application 6

1.2 An Approach to Learning Visual Parameters 7

2 General Scheme 10

3 Application Domain Constraints 12

4 Retrieval based on Similarity 13

5 Learning by Observation 15

6 Retrieval based on Conceptual Clustering 21

6.1 COBWEB 21

6.2 Enhancing COBWEB capabilities 22

7 Data Base Update 24

8 Snakes 25

9 Experimental Results: Learning and Selecting Snake Parameters 27

9.1 Presentation 27

9.2 System Evaluation 33

9.3 Evaluation of Retrieval Procedure 34

10 Conclusion 39

A Snakes 41

A.1 2–D Linear Snakes 41
A.2 3–D Linear Snakes 43
A.3 Ribbons 44

1 Introduction

The research effort described here is an attempt to make computer vision systems more effective by endowing them with a capability to learn. Our philosophy has been shaped by the following principles, each of which has motivated some aspect of our approach:

> *Learning is essential — image understanding system designers cannot anticipate all possible situations that may arise in advance.*

No matter how much effort is devoted to building and refining a knowledge base, there will always be limits to the breadth of competence provided. Even if it were possible to construct a knowledge base sufficient for supporting the entire range of anticipated tasks, the required effort and expense make it infeasible to do so in all but the most limited applications. Effective mechanisms for enabling a system to acquire its expertise over time can have a significant impact on our ability to construct systems that become more (rather than less) competent as they age.

> *A vision system should improve its performance through experience.*

Rather than analyzing images in isolation and throwing away the results, a vision system should interpret an image in the context of what it already knows about the scene. In addition, the results of its interpretation should augment its knowledge of the scene and the extraction task, and the system should be able to use that information to analyze similar situations more effectively in the future.

> *An intelligent system should never be idle.*

If an intelligent system has the ability to learn through experience, it might as well devise its own training examples to more fully exploit that ability. Rather than maintain a static knowledge base when the system is not otherwise engaged, it should concoct new situations or revisit previous ones, invoke its repertoire of reasoning or visual capabilities, and update and reorganize its knowledge base according to the results.

While the accomplishment of any of these objectives is profoundly difficult, we nevertheless have endeavored to construct a framework that allows the exploration of a particular approach to learning

that offers the promise of at least partial solutions. More specifically, our overall goal is to devise a practical computer vision system that can

- Recognize a class of objects in a set of related images
- Build an enhanced description of the environment from the sequence of images it processes
- Use its enhanced description of the environment to improve its recognition capabilities

1.1 A Cartographic Application

Imagine a cartographic application in which the photointerpreter's task is to model all roadways within a given geographic area by extracting such features from a collection of overlapping overhead images. The photointerpreter is to be assisted by a partially automated system designed to support 3–D map construction.

The traditional interactive approach embodied in today's operational systems [1] can be described as follows:

> The system has a collection of algorithms that are suitable for extracting model instances of specified objects. The user chooses the algorithm to be used to extract a particular object. A menu is provided containing the default values of parameters, which the user can override if he chooses. The algorithm with the designated parameters is then applied to the image(s) producing a resultant model instance. The user has the option to accept the result, to modify the result manually, to rerun the algorithm with a different set of parameters, or to choose a different algorithm.

Two of the problems that the user of such a system must face are the choice of algorithm and the setting of its associated parameters. These requirements mean that the user must have a fairly high degree of expertise with the algorithms to accomplish the extraction task effectively.

If, on the other hand, the system is itself able to learn how to select among its algorithms and to set their parameters through its experience with similar extraction tasks, it should be possible to reduce the need for operator expertise while improving efficiency at the same time.

[1] e.g., RCDE, GLMX, DSPW, KBVision, Geoset

The approach we propose and describe in this paper addresses the feature extraction task as follows:

> The system has a collection of algorithms that are suitable for extracting model instances of specified objects. The user chooses the class of objects to be extracted and provides information about his task and the scene being analyzed. The system compares the extraction context to its prior experience with similar extraction tasks, to choose an appropriate algorithm and parameter settings for the current task. The algorithm with the designated parameters is then applied to the image(s), producing a resultant model instance. The user then has the option to accept the result, to modify the result manually, to rerun the algorithm with a different set of parameters, to choose a different algorithm, or to ask the system to provide an alternative selection of algorithm and parameters. The system updates its database of experience with the outcome for use in subsequent extraction tasks.

The focus of this paper is on establishing a foundation for machine learning in interactive image understanding systems — how can a computer vision system use its experience to improve its competence? The design and implementation of a widely used interactive computer vision system, the RADIUS Common Development Environment (RCDE), has been discussed elsewhere [14, 23]; this effort is intended to enhance the performance of RCDE through the addition of a learning component. We build upon the notion of a context-based vision system that has also been previously developed [25, 26].

1.2 An Approach to Learning Visual Parameters

The user in our interactive system is a critical component in the automated learning process, even though he is not necessarily aware of this role. The user provides the performance evaluation and correction feedback that is necessary for directed learning and avoids the need for a computationally infeasible trial-and-error approach. Our system takes advantage of the human review of its results to determine how successful it has been, to reinforce its successes, and to take corrective action for its failures.

Two different problems must be addressed. The first one is to find the best algorithm to run, and the second one is to determine the best parameter setting for the selected algorithm in the given

context. Only a few authors have addressed these problems, and all are concerned with finding the best strategy to accomplish a complex task necessitating several steps or subtasks. At each step and depending on the data, they pick the best procedure among all possible to perform the subtask. They focus on the choice of an algorithm, and typically are unclear on how they set the control parameters. In the domain of pattern recognition, we can cite the work of Draper et al [8, 9], which involves learning the best strategy to identify a certain type of object. A very interesting point about this work is that they are able to give an upper bound of the cost of each strategy [9]. We can also cite the OKAPI system [5], which more generally, can select the best image processing chain to perform a given task. OKAPI has been applied successfully in the domain of galaxy recognition.

Our approach has been guided by two points. First, our concern is quite different from the ones above, because we are considering only one-step tasks[2], and we assume we can have, in certain cases, only one algorithm to achieve the task. Thus, we are primarily interested by the choice of good parameter settings for a given algorithm.

Second, we believe that it is useless to choose a so-called best algorithm for a given task if we are unable to set the control parameters correctly for any set of data.

Thus, the scheme we adopt is first to find the "best" parameter setting for every algorithm available to achieve a specified task. Then we give a score to each parameterized algorithm, showing the chances of the algorithm to succeed with the particular parameter setting. Finally, we run the top-ranked algorithm.

Our approach is based on the use of contextual information. Many authors have used contextual information in image understanding systems [20, 7, 4, 27], but few have made the use of context a way to improve the performance of systems through experience and learning. We define *context* as any information that may characterize the task or input data given to a vision process. Thus, image resolution is part of the contextual information, as is the camera geometry, a priori scene knowledge, and the purpose of image analysis. The approach relies on the assumption that each "running context" (or simply context), consisting in a set of contextual values, can be related to an acceptable parameter setting with a minimum of ambiguity. This requirement implies a kind of continuity in both context and parameter spaces. This continuity can be expressed as follows:

Different contexts require, in general, different parameter settings while identical or close contexts

[2]Note that this is not restrictive because if a task is more complex and requires several steps we can apply the procedure we have designed on each step of the task.

would require a single or similar parameter setting(s).

Using this hypothesis, our learning problem can be defined as a problem of generalization where the goal is to find how the use of one parameter setting for one context can be generalized to similar contexts, or more generally, nearest contexts.

In Section 2, the general scheme of the system is presented. We see that the various possibilities offered to the user make this scheme very flexible. This flexibility and the need for using various algorithms has implied certain constraints in the design of the system. We present them in Section 3. The principal problem that arises is the presence of both numerical and categorical context elements.

In Section 4 we focus on the retrieval problem which, given a new context, consists in finding the nearest contexts that are present in a data base. To find the nearest contexts, we propose a measure based on similarity between context element values. This measure deals with both numerical and categorical context elements.

This kind of approach has some computational limitations and limited learning capabilities. Thus, we have investigated another method based on incremental categorization, that places a new context into a category where nearest already encountered contexts can be found. This kind of generalization is very important to reducing the search time for similar situations. *Learning by observation*, and more particularly *conceptual clustering*, which is a type of learning by observation aimed at producing a classification for the observations, are well adapted to this categorization problem.

In Section 5, we present different aspects of learning by observation and review some existing systems using this technique. We see the extent to which the constraints of our problem are satisfied by existing systems.

In Section 6, we focus on one particular method of conceptual clustering, and we enhance this method to deal with heterogeneous types of variables.

In Section 7, we present the learning update, consisting in updating with new examples the data bases where past experiences are stored, and eventually automatically learning a correct parameter setting in cases where the system was unable to provide one. This phase is essential to improving learning capabilities of the system.

In Section 8 and the Appendix, we describe the snake algorithm that we use to demonstrate our system's effectiveness. Snakes [15, 28, 12] are a very powerful technique for edge detection that integrate information from both photometric and geometric models in an optimization framework.

More specifically, we show how our implementation allows the various parameters to be context-specific as opposed to image-specific.

Finally, in Section 9, we present some experimental results; we show how our system makes the use of vision algorithms easier by reducing the required user expertise while improving his efficiency. We have implemented and tested an initial design and demonstrated successful performance. Our experiments involved the extraction of 68 features in four images of two different sites. These results are presented in graphical form where the effects of successful learning are clearly apparent.

2 General Scheme

The system we have designed is intended to use computer vision algorithms to extract cartographic features from a set of imagery under human guidance. The system makes use of a collection of algorithms, and must select appropriate parameter values prior to each invocation of an algorithm. Choice of algorithm and parameter settings is to be made on the basis of contextual information.

In the following a *context element* defines a contextual variable. A *context* will be a generic term to define a contextual environment. Each context is characterized by a context vector regrouping context element values. A *parameter setting* is a set of parameter values needed to run an algorithm. A parameter setting is more formally represented by a *parameter vector*.

Let $\mathcal{A}(\mathbf{D}, \mathbf{P})$ be a process that takes two kinds of information as input, data represented by a vector $\mathbf{D}$ (which includes specification of the task), and parameters represented by a vector $\mathbf{P}$. $\mathcal{A}$ gives a solution $\mathcal{S}$ as output. Suppose we have calculated some context information about the task and data represented by a context vector $\mathbf{C}$.

The primary performance criterion for a practical system for interactive feature extraction is its ability to generate a good result in a previously encountered situation while avoiding the repetition of past errors. Thus, it is necessary for the system to keep a record of its successes and failures. Operator review of the automatically generated results provides the necessary information in a natural way.

From these considerations, we derive the general scheme for our system, depicted in Figure 1. One data base (DB) is associated with each algorithm available to perform a given task. These data bases contain past experiences expressed by pairs ($\mathbf{C}$, $\mathbf{P}$) given the correct parameter setting $\mathbf{P}$ in the contextual situation $\mathbf{C}$. Given a new context vector $\mathbf{C}$, these data bases are used to retrieve the

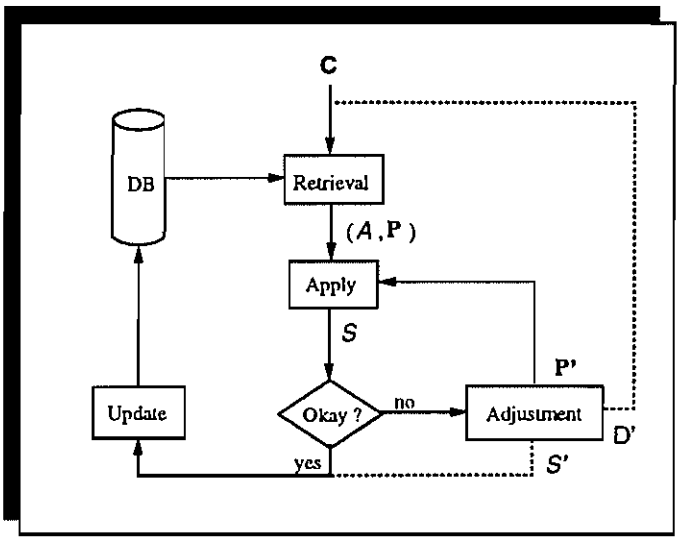


Figure 1: General scheme

"best" algorithm $\mathcal{A}$ and a parameter vector $\mathbf{P}$. The process $\mathcal{A}(\mathbf{D}, \mathbf{P})$ is applied and the user checks the validity of the result S. If S is acceptable, $\mathbf{P}$ is stored in order to update the parameter value probabilities in the data base. If $\mathcal{A}$ failed, the user has three adjustment options. The first one is to manually modify the parameters and to apply $\mathcal{A}$ again until it succeeds. The second option is to manually modify the result S so it becomes reasonably good. In this case, updating the data base consists in learning a correct parameter setting for the current situation. Because the solution S is available, the learning can be done automatically. This phase is usually performed after the session. Finally, the third option for the user is to modify some aspects of the input data $\mathbf{D}$. In such a case, a new context vector is calculated and the procedure starts all over again. In this way, the system is able to improve its performance over time, as the likelihood that the system encounters a context similar to one in which it has successfully accomplished its task increases monotonically.

This architecture can serve as the foundation of a practical system for cartographic feature extraction, while affording a path to the creation of a vision system with very powerful constructs for learning.

3 Application Domain Constraints

The interactive nature of our system and the wide variety of data types it must deal with pose additional challenges to the design.
The most important constraint (or capability) we have is to be able to deal with different types of data. Both context elements and parameters can be either numerical or categorical. Table 1 shows examples of some context elements that can be used. The *resolution* is a numerical context element with continuous values. On the contrary, the *task* has categorical (nominal) values. Finally, *desired accuracy* is a context with ordinal values – that is, symbolic values that can be related to a numerical discrete function (like $low = 1$, $middle = 2$, $high = 3$).

context element	type	possible values
resolution	numerical, continuous	0.1, 10, 100
task	categorical	road, building delineation
desired accuracy	ordinal	low, middle, high

Table 1: Different types of variables handled by the system.

Most of the existing systems found in the literature consider only one type of data. Few systems use both numerical and categorical variables, but consider numerical values as categorical. This is often acceptable with ordinal variables, but not with continuous numerical ones.

Employing numerical values is a major problem for generalization, because close values cannot be considered the same as completely different values. On the other hand, dealing with categorical variables can also be a real problem when similarities between values are required. We will see in Sections 4 and 6 two solutions to deal with all types of variables presented in Table 1.

A second constraint we have about variables is due to the fact that our system is interactive. One role of the user is to provide values for context elements that cannot be computed automatically. The interactivity is an opportunity to increase system performance. However, the user must not be annoyed with too many constraints. Thus, we have to take into account that the user can decide not to provide every context element values, and so that some context elements may have the value *"unknown"*.

The number of context elements we use as well as the number of parameters an algorithm has can be high. For instance, results shown later involve sixteen context elements and an algorithm

with eight parameters. Spaces of context elements and parameters have a high dimension, and the expertise of the system cannot cover these spaces entirely. There are two consequences. First, the system should be able to learn in an incremental way. Second, because of high dimensions, initial learning may be poor and the learning method has to be able to deal with radical changes.

4 Retrieval based on Similarity

Here, we focus on the retrieval module of our design as depicted in Figure 1. Given the context of a feature extraction task, we wish to identify the previously encountered contexts that are "nearest" to it. The parameter settings associated with those nearest contexts will be used to choose the parameters for the current extraction task.

Using our hypothesis of continuity in context and parameter spaces, and given a new context vector $\mathbf{C}$, the retrieval problem can be more precisely defined as finding the nearest context vectors $\mathbf{C}_{j1},..,\mathbf{C}_{jn}$ present in the data base of algorithm $\mathcal{A}_j$, and providing parameter vectors $\mathbf{P}_{j1},..,\mathbf{P}_{jn}$ associated with these nearest context vectors. A score has to be associated with each $\mathbf{P}_{ji}$ to rank the parameter setting according to its ability to produce a good result. The top-ranked pair ($\mathcal{A}_j$, $\mathbf{P}_{ji}$) is finally provided to the user as having the maximum chance to succeed as it corresponds to the nearest situation already encountered[3].

We investigate a measure based on similarity to find the nearest context vectors $\mathbf{C}_{j1},..,\mathbf{C}_{jn}$ present in a data base. Information retrieval is a domain that has produced a large number of similarity measures to comparing two vectors (see [29] for an extended list). Let $\mathbf{u}$ and $\mathbf{v}$ be two vectors in a n dimensional space. An important family of pseudo linear similarity measures between $\mathbf{u}$ and $\mathbf{v}$ is defined by:

$$m(\mathbf{u},\mathbf{v}) = \frac{\mathbf{u}.\mathbf{v}}{N_1(\mathbf{u})N_2(\mathbf{v})} \tag{1}$$

where $\mathbf{u}.\mathbf{v} = \sum_{k=1}^{n} u_k v_k$ denotes the scalar product between $\mathbf{u}$ and $\mathbf{v}$, and N_1 and N_2 are two normalizing functions (generally $N_1 = N_2$). For example, the very popular *cosine measure* is defined with $N_1(\mathbf{u}) = \|\mathbf{u}\|$, and $N_2(\mathbf{v}) = \|\mathbf{v}\|$, where $\|.\|$ denote the l_2 or Euclidean norm.

[3]In general, a few pairs will be retrieved at the same time. It offers the user the possibility to choose the one he considers the best.

This kind of ranking guarantees *under certain conditions*, that the less preferred objects are not ranked ahead of the preferred [29]. However, because of the possible categorical context elements in our problem formulation, this kind of interesting ranking function is not directly usable.

A heuristic solution to this problem is to use one of these ranking functions in a real valued space, transformed from the context space. The transformation we apply can be viewed as a normalization of context element values. We define a transform function $T_C : C \rightarrow [0, 1]$, $c \rightarrow T_C(c)$. If we consider a context C which value is c, $T_C(c)$ is the normalized value of C. To better understand what kind of normalization we are performing, and by analogy to fuzzy logic, let us associate a predicate to each context element. T_C is going to quantify how much the value of C supports the predicate associated with C.

Table 2 shows the result of the transform for different kinds of context element. `Look Angle` is a numerical context, taking its value between 0 and 90. In this particular case, we have $T_C(c) = 1 - c/90$. For example, if `Look Angle = 0`, the context value fully supports the predicate "Look Angle is nadir", and so $T_C(0) = 1$. `Sensor Type` is a categorical context. In this case, $T_C(c)$ is one if $c =$ `Visual`, and zero otherwise. With this kind of transform, all differences between two categorical elements are considered identically (i.e., zero, the maximum of disagreement). Finally, `Desired Accuracy` is a numerical discrete context element. In this case T_C is a discrete step function (with three steps in this particular example).

Context Element	Context Type	Predicate	c_1	c_2	$T_C(c_1)$	$T_C(c_2)$
Look Angle	Numeric Continuous	"Look Angle is nadir"	0°	45°	1	0.5
Sensor Type	Categorical	"Sensor type is Visual"	Visual	Radar	1	0
Desired Accuracy	Numerical Discrete	"Desired Acuracy is high"	middle	low	0.5	0

Table 2: Exemple of normalization performed on three types of context element.

In the transformed context space, we use a similarity measure defined by Equation 1 to rank nearest context vectors. Normalization (N_1 and N_2 functions) is required when we want to keep the length of the vector from being taken into account in the similarity measure. This is particularly interesting for parallel vectors with different lengths. However, because vector norms and parallelism have no special meaning in our problem, we have chosen $N_1(\mathbf{u}) = N_2(\mathbf{v}) = 1$.

Thus, given the new context vector $\mathbf{C} = (c_1, .., c_n)$, and one context vector $\mathbf{C}_i = (c_{i,1}, .., c_{i,n})$ of a data base, we define the similarity S between $\mathbf{C}$ and $\mathbf{C}_i$ by the following inner product:

$$S(\mathbf{C}, \mathbf{C}_i) = \sum_{k=1}^{n} T_k(c_k) T_k(c_{i,k}) \tag{2}$$

where T_k is the transform function associated with context element k. Let n_1 be the number of parameter settings provided to the user. The n_1 highest similarity values provide the nearest context vectors present in a data base and, so, n_1 parameter settings associated with these context vectors. If we perform this ranking for every algorithm available, the n_1 overall highest similarities provide the n_1 best algorithm and parameter settings that can be proposed to the user.

As we will see in Section 9, the performance of this method is reasonably good. However, if the number of context vectors present in a data base is too large, it may take too long to use this method in an interactive scheme. Moreover, the method provides very limited learning capabilities.

Thus, in the next sections we present another method based on the notion of learning by observation, and more specifically based on conceptual clustering.

5 Learning by Observation

The main goal in *learning by observation* is to build a representation of concepts supported by examples presented to the system. Generally, two problems must be addressed in learning by observation. The first one is to identify relevant concepts from the examples, and the second one is to find an appropriate representation of these concepts. Here, we describe some systems that learn by observation, and we focus on the two problems of concept formation and concept representation.

BACON [17] is a domain-specific system dedicated to chemical concept (law) formation. Concepts are represented by mathematical equations. Given a general a priori form of equation, BACON searches through the space of data, and the space of laws, to discover relations between data variables. If the way of representing concepts is natural in BACON, this is not the case for general-purpose systems of learning by observation. It is important to see that the choice of a particular representation is essential and dictates what concept a system is going to be able to learn.

The UNIMEM system [18, 19] uses a technique called Generalization Based Memory (GBM) that stores examples in a hierarchical data base (memory) describing concepts with increasing specificity. Concepts (generalizations) are represented by nodes in a discrimination net. Each node is a set of examples supporting the concept of the node. Learning is incremental. Each time a new example

is considered, a controlled search is performed in the GBM to find the most specific concept(s) that describe the new example. During the search process, the matching of a new example to a generalization is based on a numerical measure of similarity between values of the new example, and values of a generalization [19].

In the BLIP system [30], both knowledge and concepts are represented by rules and predicates. To try to minimize the bias introduced by the representation mode of knowledge, knowledge rules are translated into domain-independent meta rules. These meta rules are used to generate new rules (concepts) and meta rules. The generation of new concepts is *demand driven* – that is, it is triggered by the weakness (lack of information) of the current representation of knowledge.

Both BACON and BLIP employ *constructive learning.* In the process of concept formation they create descriptors not present in the input data. This creation is very important in the learning process.

In our particular problem of finding a parameter setting for an algorithm in a given situation, we could think of expressing the expertise to determine a correct parameter setting by a set of rules as in the BLIP system – that is,

In situation 1, parameter A has to be increased by 0.1 from its standard value

However, this kind of expertise has major limitations, because many counter examples are going to be found to rules like the one above. Moreover, it is known that systems employing this kind of expertise perform poorly in *complex domains.* Studies have shaken the theory of learning involving general rule sets and imply that human expertise is based on the ability to compare a current situation to previous ones [10]. This theory is supported by neural network implementations that show that it is possible to design expert systems that do not reason (in the sense of manipulating rules) but rather act using similarity with past experiences [13].

A particular type of learning by observation is called *Learning from examples.* Both types of learning have the same goal, the determination of concept generalization. However, in the case of learning from examples, concepts are created for a particular purpose: the classification of examples. This implies new constraints for existing concepts. They have to form a partition in the space of examples – each example must verify or refine only one concept.

There are two types of learning from examples. The first one, called *concept learning*, assumes that a teacher is available to preclassify examples. The system has to produce a description of the

examples of each class, which is general enough to accommodate every example, but discriminating enough to avoid interclass ambiguity. Among existing systems using the learning-from-examples approach, we cite INDUCE1.1 [21] and the system introduced by Cromwell and Kak [6]. Both systems use conjunctions of predicates to represent concepts, and both manipulate four types of data: nominal (categorical), ordinal (numerical discrete), numerical (continuous), and hierarchical. The Cromwell and Kak system includes more kinds of generalization rules, and is applied to a real case of pattern recognition in the domain of electronic component recognition.

When a system does not need preclassified examples, but rather is able to determine by itself the relevant classes (concepts) to create, we talk about *concept clustering*. The classification problem is then generally performed in two steps. The first consists in determining appropriate clusters, and the second one consists in characterizing clusters (as in concept learning). The main problem is to determine a measure to evaluate the quality of the clustering. Several criteria have been tested. Systems like CLUSTER/2 [22] and CLUSTER/S [24] use the notion of simplicity (comprehensibility) of the concept representation. Unlike statistical clustering techniques (based on numerical similarities) that produce clusters difficult to analyze, they make the assumption that the simpler the class description, the better. To avoid having a trivial partition with all examples in the same class, the final description must closely match the original examples. This clustering criterion relies on a psychological study showing that people who are asked to classify complex data choose only one or a few simple features from the data to build a set of disjoint classes. CLUSTER/2 and CLUSTER/S use conjunction of predicates to represent concepts. In addition, CLUSTER/S uses a priori knowledge related to the application domain (expressed with a semantic net called Goal Dependency Network) to determine the relevant descriptive predicates for a given goal, and thus guide the concept formation.

Kodratoff and Tecuci [16] use a *conceptual distance* as a criterion for cluster formation. Examples are represented using conjunction of predicates. The defined conceptual distance is based on the idea that two very different examples are generalized in an expression very different from the original examples (in terms of predicates, arguments of predicates, and number of predicates in the expression), while similar examples can be generalized to themselves. Thus, both the generalization and the process of obtaining this generalization indicate the conceptual distance between examples.

A third criterion used to evaluate cluster quality consists in maximizing the inference ability of the resulting partition. The idea is that the better you can predict features based on class membership, the more advantageous it is to create such a class. Both COBWEB [11] and the system presented by Anderson [2, 3] use this kind of criterion in an incremental learning scheme. A new example is

placed in the class that maximizes the predictability of the category formed by the addition of the new example, or is classified in a new category if adding the example to an existing category does not improve predictability of any of the classes. The number of classes is determined automatically by the system. These incremental schemes are interesting because learning is easily accomplished by considering new examples. However, the resulting clustering is sensitive to the order in which new examples are presented to the system. To minimize this dependency, as new examples are added, COBWEB tries to split or merge classes to improve the partitioning.

Table 3 presents a summary of important features of the learning systems described above. These features are:

- *Learning type*: Conceptual clustering is the most appropriate method for our problem. It does not require a teacher giving preclassified examples, and so reduces to a minimum the need for human expertise. It also produces a partition of the space of examples.
- *Concept formation*: similarity-based models, predictability-driven models, and maximizing comprehensibility models are fairly similar. In general, predictability-driven methods will be sensitive to the number of features shared among objects and, therefore, they tend to make the same predictions as similarity-based models [1]. The predictability-based quality measure used by COBWEB can be viewed as a continuously valued analog of the quality measure used in CLUSTER/2 based on comprehensibility [11].
- *Representation mode*: As mentioned earlier an appropriate representation of concepts is essential to be able to express all expected concepts. Representation of concepts using conjunction of predicates is valuable for its ability to manipulate heterogeneous types of data. However, no method of conceptual clustering, using conjunction of predicates, is effective when dealing with numerical variables (in contrast with categorical ones). Hierarchical structures (trees) are also interesting because, associated with some controlled heuristic search, they provide faster methods of categorization.
- *Constructive learning*: This feature remains a challenge for most existing systems. For us, it consists in learning a new similarity relation between context elements, and ultimately new context elements.
- *Data manipulated*: As mentioned previously, we need to manipulate three kinds of data: nominal (categorical), ordinal (numerical discrete), and continuous numerical. Only learning-from-example systems, and Anderson's systems use all these types of data simultaneously.

Criteria	INDUCE Michalski 80	Cromwel & Kak 91	CLUSTER/2 Michalsi, Stepp 83	CLUSTER/S Michalsi, Stepp 86	COBWEB Fisher 87	Anderson 90	Kodratoff & Tecuci 88	UNIMEM Lebowitz 83	BACON.6 Langley et al 86	BLIP Wrobel 89
Learning type	Concept learning	Concept learning	Conceptual clustering	Conceptual clustering	Conceptual clustering	Conceptual clustering	Conceptual clustering	Concept formation	Concept formation	Concept formation
Concept formation based on	-	-	Maximizing Comprensibility	Maximizing Comprensibility	Predictability	Predictability	Conceptual Distance	Similarity	-	Demand Driven
Representation mode	conj. of predicates	conj. of predicates	conj. of predicates	conj. of predicates	Probabilistic tree	Categories	conj. of predicates	Discriminant net	Mathematical equations	rules
Constructive induction	limited	no	no	no	no	no	no	no	yes	yes
Data manipulated	Nominal Ordinal Hierarchical Numerical	Nominal Ordinal Hierarchical Numerical	Nominal Ordinal Hierarchical	Stuctured objects	Nominal	Nominal Numerical	Nominal Hierarchical	Numerical	Numerical	Nominal
Incremental learning capability	no	no	no	no	yes	yes	no	yes	no	yes
Teaching required	yes	yes	no	no	no	no	no	no	no	no
Example order and concepts are dependent	no	no	no	no	yes	yes	no	no	no	no
Number of categories determined automatically	no	no	no/yes[4]	no/yes[4]	yes	yes	yes	-	-	-
Concepts produce a partition	yes	yes	yes	yes	yes	yes	yes	no	-	-

Table 3: Summary of selected system features.

[4]The number of categories is not determined automatically. However, by dividing the example classes until a manually determined bounding number of classes is attained, the system can a posteriori find what is the "best" number of classes.

- *Incremental learning capability*: In the domain of conceptual clustering, only Anderson's system and COBWEB provide an incremental scheme required in our problem formulation.
- *Teaching required*: Because we want to make the use of new algorithms easy, we cannot use a teacher for training. The role of a human operator using our system must only be to evaluate the algorithm results.
- *Example order and concepts are dependent*: Incremental schemes of conceptual clustering are computationally effective, but sensitive to the order in which examples are presented to the system. COBWEB uses a technique of split and merge to reduce this effect. Anderson [3] states that though the category structure can vary substantially as a function of order, the predictions delivered by the different categories do not differ much themselves.
- *Number of categories are determined automatically*: Because we have no teacher, this feature is essential.
- *Concepts produce a partition*: This feature is preferred to avoid ambiguities, but is not strictly required, if examples appear in only a few categories.

From this evaluation, we can see that the most appropriate method of learning seems to be the conceptual clustering approach using an incremental scheme of clustering and a hierarchical representation of concepts. However, no systems having these characteristics and the ability of dealing with both categorical and numerical variables already exists. Only two systems use conceptual clustering in an incremental scheme: Anderson's system and COBWEB.

Anderson's system is not hierarchical, and has the following major inconveniences. (1) It uses a Bayesian approach relying on strong assumptions about independency of probability distributions, and form of distributions of the manipulated variables. (2) A priori knowledge that we may not have for every algorithm we use is required to set some of the several parameters of the system. (3) The number of different values of a categorical context element has to be known a priori. (4) Finally, as we will see in Section 9, parameters of the system seem to be very sensitive and hard to set correctly. Anderson's approach is concerned with deducing psychological conclusions from the parameters associated with the best (or expected) categorization – parameter values are found just by doing experiments. Furthermore, it seems that no standard values exist; parameter values are really application-dependent.

So, even if COBWEB does not deal with continuous numerical variables, it seems to be the most adapted to our problem, and it is possible to enhance COBWEB capabilities to take into account numerical variables.

6 Retrieval based on Conceptual Clustering

Because of the number of context vectors that can be present in the data base, we cannot afford an exhaustive search for the nearest contexts. By categorizing context elements, we reduce the amount of search, because as a new context is presented to the system, it can be compared to each category instead of each context. A hierarchical organization of categories allows the search process to be even faster.

Thus, the incremental conceptual clustering scheme using a hierarchical representation of concepts seems well adapted to our problem. A conceptual clustering method of categorization can be used as a search method to categorize a new context vector C. The resulting category can be considered as containing the contexts nearest to C. The conceptual clustering quality measure after categorization of the new context vector can be used to rank the algorithms available.

As mentioned earlier, COBWEB [11] is the only system using incremental conceptual clustering that is well-adapted to our problem. It seems now necessary to present in more details COBWEB, focusing on the quality measure of clustering and the search (categorization) process of this system.

6.1 COBWEB

In COBWEB, examples are represented using attribute-value pairs. Values are only nominal or ordinal. Concept formation is based on predictability. The quality of a clustering partition $\{\omega_1, .., \omega_n\}$ is measured using the following *category utility*:

$$U = \frac{\sum_{k=1}^{n} P(\omega_k) \sum_i u_{ik}}{n} \tag{3}$$

with

$$u_{ik} = \sum_j P(C_i = V_{ij}|\omega_k)^2 - \sum_j P(C_i = V_{ij})^2 \tag{4}$$

where n denotes the number of classes in the partition. $P(C_i = V_{ij}|\omega_k)$ is the probability for context element C_i to have value V_{ij} in class ω_k, and $P(C_i = V_{ij})$ is the probability for context element C_i to have value V_{ij} over the whole partition. U measures the increase in the expected number of context element values that can be correctly guessed knowing the partition $\{\omega_1, .., \omega_n\}$ over the expected number of correct guesses when no partitioning is given [11]. As we can see, Equations 3 and 4 are fully applicable only for categorical, and eventually for ordinal values.

The search structure (called Control Structure) used to categorize a new example in COBWEB is presented in Table 4. At each level of the classification tree, the utility of the creation of a new class in the partition is compared to the one of the insertion of the new example in an existing class. The search is a recursive descent into the tree of categories. At each level, the best host for a new object is defined as the class having the maximum category utility (as calculated with Equations 3 and 4) after having added the new example. Splitting or merging is attempted on best hosts to reduce the effect of example order.

Function COBWEB(Object, Root)
1) Update probabilities of the root
2) If Root is a leaf
THEN return the expanded leaf to accommodate the new object
ELSE find that child of Root that best hosts Object and perform one of the following
a) Consider creating a new class and do so if appropriate
b) Consider merging two best hosts and do so if appropriate and call COBWEB(Object, Merged node)
c) Consider splitting best host and do so if appropriate and call COBWEB(Object, Root)
d) call COBWEB(Object, Best child of Root)

Table 4: Control Structure of COBWEB. From [11]

6.2 Enhancing COBWEB capabilities

Fisher's idea dictates that the category utility, expressed by Equation 4, is a measure of the increase of expectation of a value knowing a partioning, over the one when no partitioning is given. For numeric values the expectation of a value is measured by the variance of the value distribution. The narrower the distribution, the smaller the variance, and the better a value from the distribution can be predicted. Thus, for numerical values, we suggest the following term to take place in the category utility measure:

$$u_{ik} = \frac{s_i^2 - s_{i\omega_k}^2}{s_i^2 + 1} \tag{5}$$

where $s_{i\omega_k}^2$ is the variance of the distribution of values of the i'st context element in class ω_k, and s_i^2 is the variance of the distribution over all the partitions. Adding one to the denominator is necessary to avoid problems with null variances. If a numerical context element is identically distributed on a class ω_k and on the whole partition, the context element is irrelevant in any partitions, and we have $s_{i\omega_k} = s_i$ and $u_{ik} = 0$

	Var1	Var2	Var3
Cat1	3	5000	60
Cat2	2	1000	100
Cat3	3	2000	150

Table 5: Example of categorization with three variables and three categories.

To demonstrate the categorization ability of our measure, let us consider the following simple example. Table 5 shows a categorization example with three categories and three variables. Variables are centered on the values indicated in the table. From this definition we have created 15 examples (5 from each category) by adding gaussian noise on central values. Resulting examples have been presented *randomly* for categorization, using COBWEB search structure and the category utility measure based on Equation 5. The final tree we obtain is presented in Table 6.

As we can see, the system was able to rediscover the three categories (`Class 1`, `Class 3`, and `Class 4`). Since Category 2 is nearer to Category 1 by considering the two last variables, it is comforting to see that we find this relation in the tree. In all experiments performed using this example we always found the tree shown in Table 6. Thus, the search structure, designed to minimize the effect of example order, seems to perform reasonably well even with numerical values. Results of different retrieval methods are presented in Section 9.

Since a new context has been categorized, parameters associated with nearest contexts can be retrieved. After being categorized, the new context becomes a leaf of the category tree. Near categories (brothers) in the tree represent nearest contexts from the new context.

However, the structure of the category tree can vary locally. It may be deep at some places, and not in others. As a result, the number of brothers a leaf has can be very different from place to place.

```
---> Class  0 ---> Class  2 ---> Class  4 ---> Class 25 : Ex = (4 5558 58)
                                 |------> Class  8 : Ex = (3 4939 62)
                                 |------> Class  7 ---> Class 16 ---> Class 18 : Ex = (3 5181 52)
                                                                |------> Class 17 : Ex = (3 5164 53)
                                          |------> Class 15 : Ex = (2 5135 48)

                    |------> Class  3 ---> Class 14 : Ex = (1 9506 100)
                                 |------> Class  6 : Ex = (2 10497 103)
                                 |------> Class  5 ---> Class 13 : Ex = (2 10100 106)
                                                |------> Class 12 ---> Class 24 : Ex = (2 9058 111)
                                                                |------> Class 23 : Ex = (3 9947 110)

          |------> Class  1 ---> Class 11 : Ex = (0 19230 141)
                       |------> Class 10 ---> Class 22 : Ex = (1 19718 147)
                                      |------> Class 21 : Ex = (1 19810 150)
                       |------> Class  9 ---> Class 20 : Ex = (2 20439 146)
                                      |------> Class 19 : Ex = (1 20618 148)
```

Table 6: Example of categorization with three variables and three categories.

Thus, the generalization may be sometimes too wide and sometimes not wide enough. To avoid this problem, and be able to always provide the same number n_1 of acceptable parameter settings, we adopt the following strategy. From the new category, we climb in the tree to pick surrounding leaf categories until a number n_0 of categories is reached ($n_0 > n_1$). Then we sort the contexts of each category depending on their similarities from the new context (nearest context first) using the similarity measure expressed in Section 4 by Equation 2. Parameter settings associated with the n_1'th first contexts are provided to the user.

This method has several advantages over the one presented in Section 4. First, it may be faster to retrieve a correct parameter setting when data bases contain a large number of examples. Second, the predictability measure performs well when lots of examples are involved. During categorization, when we approach the leaves of a tree, performance decreases. By using a similarity measure over the n_0 nearest contexts, we greatly improve final performance. Finally, the tree structure gives very useful information about contexts and parameters, and particularly their discriminatory ability with regard to contextual situations.

7 Data Base Update

Updating the data bases is necessary for the system to improve its performance through experience. This process is performed at the end of a session; thus, update running time is not important. During

the session, update data are accumulated in a file. This file contains the successes as well as the failures of the system to provide correct parameters. Every attempt to provide a parameter vector is analyzed (see Figure 1). If the attempt was a success – that is, if the solution $\mathcal{S}$ returned by the process $\mathcal{A}(\mathbf{D}, \mathbf{P})$ was really the one expected by the user – the automatically retrieved parameter(s), and the manually set parameters (in the case where the user had to set $\mathbf{P}$ manually), are incorporated into the data base, and associated probabilities are adjusted.

If an attempt resulted in a failure – that is, if the user had to manually indicate the solution $\mathcal{S}$ he wanted – then a search process is run to find the parameter vector that best reproduces the given correct solution. The context vector and the best parameter vector(s) are then added to the data base.

Any solution S is stored during a session, even if the attempt was a success. It allows the same search process to be used to learn correct parameter settings *for every algorithm available*, even if an algorithm was not chosen as appropriate. Like this, learning about all available algorithms improves at the same time.

Because the data base update is performed off-line, it can be performed continuously whenever the system is not otherwise engaged. We have not yet implemented it, but our design allows for this time to be spent finding new context elements that better resolve the selection of algorithms and parameters. This facility would constitute a very powerful capacity for the discovery of new concepts – a challenging problem in machine learning.

8 Snakes

The automated procedure for parameter setting that we have described is, in theory, suitable for setting the parameters of virtually any algorithm. For purposes of evaluation, we have performed our experimentation using one class of feature extraction algorithms — an optimization approach known as snakes.

Snakes were originated by Terzopoulos, Kass, and Witkin [15, 28] and have since given rise to a large body of literature. In the original implementation, the parameters were chosen interactively, and potentially had to be changed from image to image. In our own implementation [12], which is further described in the Appendix, those parameters are computed automatically and become amenable to context-specific setting.

A 2–D snake is treated as a polygonal curve $\mathcal{C}$ defined by a set S containing n equidistant vertices

$$S = \{(x_i\ y_i),\ i = 1, \ldots, n\}$$

that can deform itself to optimize an objective function $\mathcal{E}(\mathcal{C})$.

Formally, we can write the energy $\mathcal{E}(\mathcal{C})$ that the snake minimizes as a weighted sum of the form

$$\mathcal{E}(\mathcal{C}) = \sum_i \lambda_i \mathcal{E}_i(\mathcal{C})$$

where the magnitudes of the $\mathcal{E}_i$ depend on the specific radiometry and geometry of the particular scene under consideration and are not necessarily commensurate. To determine the values of the λ_i weights in a context-specific way as opposed to an image-specific one, we have found it necessary to normalize out those influences. The dynamics of the optimization are controlled by the gradient of the objective function (Appendix, Equation A-6). We have therefore found that an effective way to achieve this result is to specify a set of normalized weights λ_i' such that

$$\sum_{1 \le i \le n} \lambda_i' = 1\ .$$

The λ_i' define the relative influences of the various components, and we use them to compute the λ_i as follows:

$$\lambda_i = \frac{\lambda_i'}{\| \vec{\nabla} \mathcal{E}_i(\mathcal{S}^0) \|}$$

where $\mathcal{S}^0$ is the estimate at the start of each optimization step. In this way we ensure that the contribution of each $\mathcal{E}_i$ term is roughly proportional to the corresponding λ_i' independently of the specific image or curve being considered.

Table 7 lists the parameters of the snake algorithm that we use to test the learning capability of our system. In practice, there are a few more, like those that define the rate of increase of the viscosity or the stopping conditions. However, since the algorithm is not very sensitive to these, we simply fix them once and for all. The categorical parameters determine the type of snake to be used, the presence or absence of a smoothing term, the optimization procedure to be used in the absence of a smoothing term, and whether or not the endpoints of the snake ought to be fixed.

Categorical parameters	
Type of snake	Snakes can model smooth, polygonal or ribbon curves.
Fixed endpoints	The endpoints of the snake can be either fixed or not.
Numerical parameters	
Gaussian smoothing	Size of the gaussian mask used to compute image gradients
Initial step size	Δ_p, pixel step size of Equation A-7 used to compute the initial viscosity
Stick length	Initial intervertex spacing of the snake, in pixels
Smoothness constraint	λ'_D weight of the deformation component, Equation A-4
Width constraint	λ'_W weight of the width component, Equation A-9
Curvature/tension ratio	Relative contribution of tension and curvature, Equations A-4 and A-8

Table 7: Snake categorical and numerical control parameters. These parameters and related equations are defined in the Appendix.

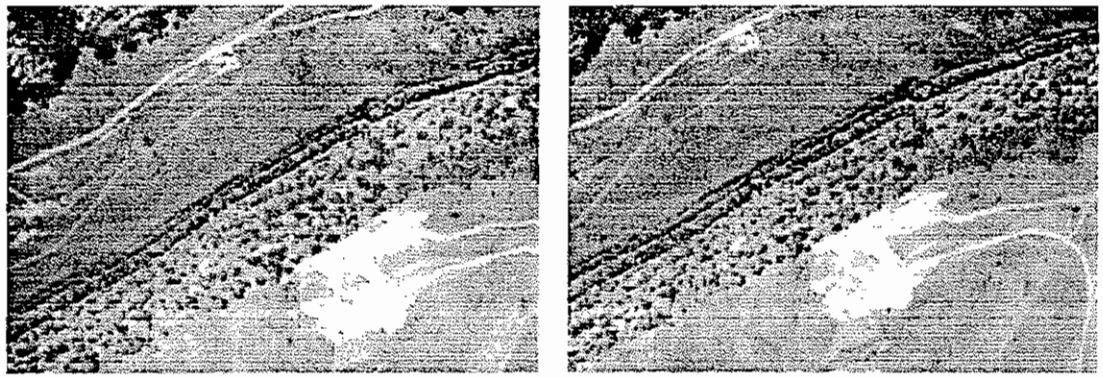
9 Experimental Results: Learning and Selecting Snake Parameters

We have applied our approach to learning the snake algorithm parameters described in Table 7. Our implementation makes use of the RADIUS Common Development Environment (RCDE) [23].

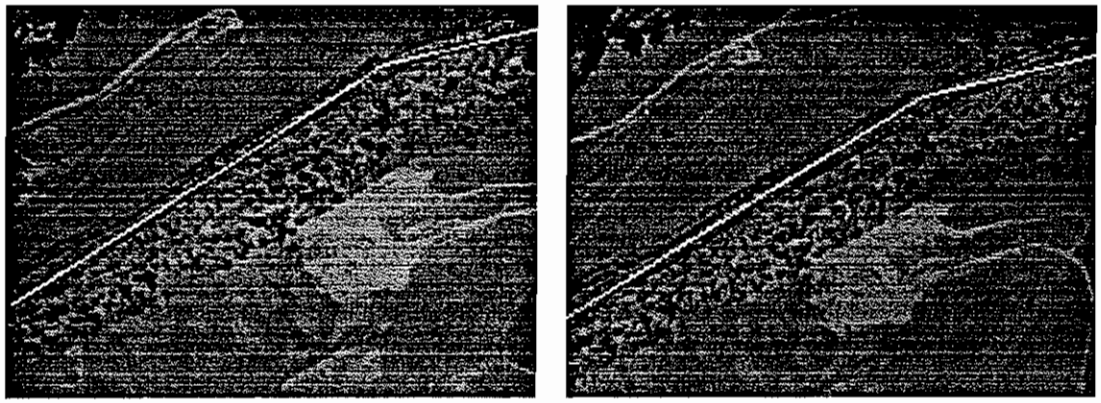
9.1 Presentation

First, let us illustrate the general scheme of our system on the following example. Suppose the user's task is to delineate the ridge present in the two images depicted in Figure 2a. The user sketches the 3–D curve in the left image of Figure 2b. The camera models and digital terrain model associated with the image site are used to draw the curve in the right image of (Figure 2b).

Then the user reviews contextual information (Figure 3). There are two categories, corresponding respectively to global image context elements and curve-specific context elements. We have eight global context elements: *Look Angle* giving the look angle of the sensor, *GSD* giving the resolution, *Sensor* indicating the type of the sensor, *Element type*, *Site type*, *Season Characteristics* that point out some season particularity (snow or rain), *Illumination*, and *Sun Angle* which is important for predicting shadows. We also have eight context elements for characterizing local contextual information: *Task* indicating the class of object to be extracted, *Seed accuracy* indicating the distance between the seed curve and the expected solution, *Desired accuracy* indicating whether or not the



(a)



(b)

Figure 2: (a) Two images from one site used in our tests. (b) 3–D seed curve defined in two views.

user wants the optimized curve to strictly follow the contours of the image, *Site type*, *Material type*, *Terrain elevation*, *Seed min angle* which is the smallest computed angle between two consecutive lines defining the seed curve (minimum local curvature), and *Gradient mean* which is the average of the intensity gradient around the seed curve.

These sixteen context elements form the context vector C. Most of the items are calculated automatically. The user can choose to not provide every item. The Parameter Selection button returns a selection of parameters based on the nearest context vector present in the data base. The user can select one of the provided parameter sets and invoke the algorithm. If the user can't find any parameter sets giving an acceptable optimized curve, he can adjust the solution manually, set

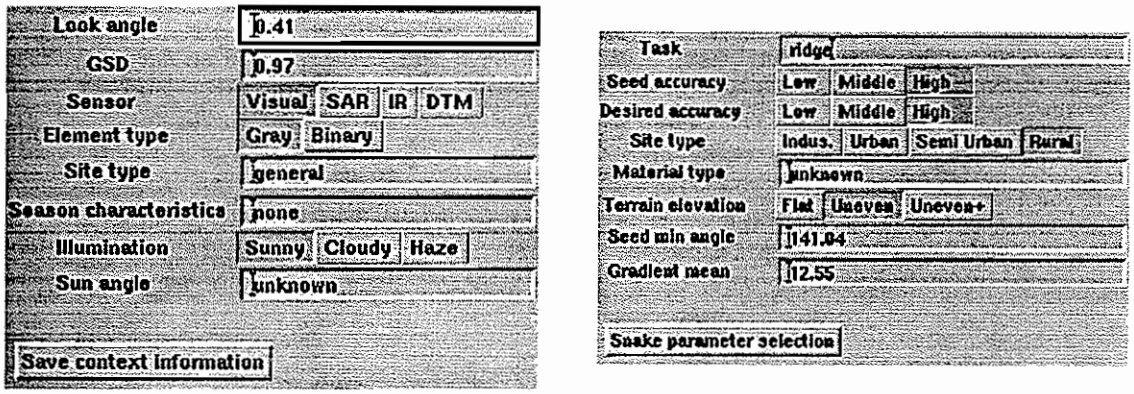


Figure 3: Context menus: global context elements (left), site-specific context elements (right).

his own parameters, or modify the initial seed curve (thereby setting a new context). When the user finds an acceptable solution, the initial curve, optimized curve, and parameters are automatically saved to update the data base. The optimized curve is presented in Figure 4.

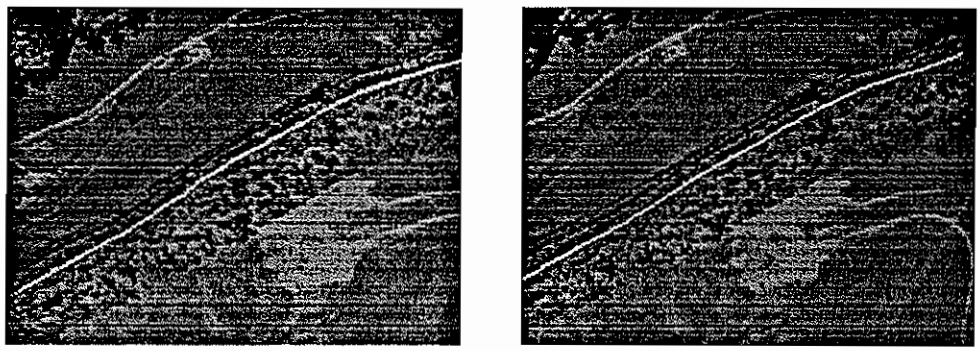
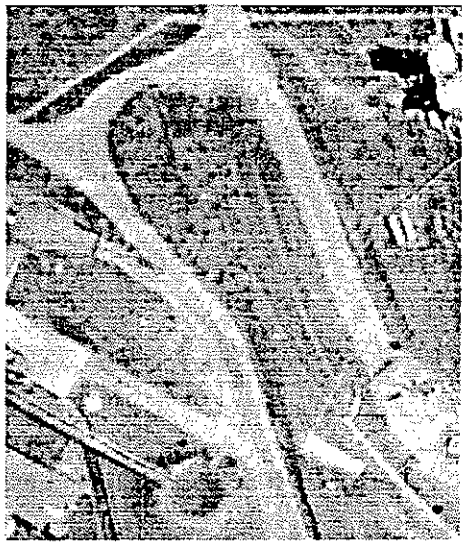


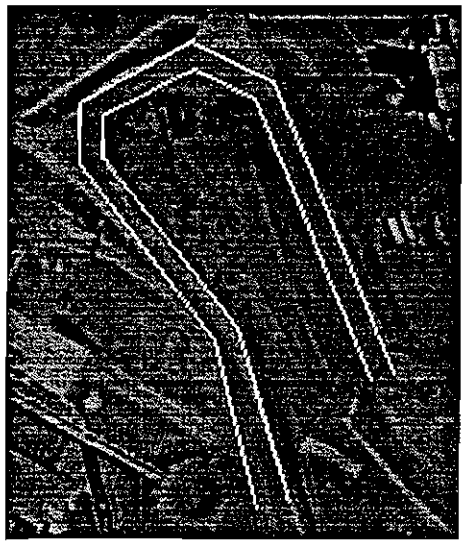
Figure 4: Snake-optimized 3–D curve

Figures 5 and 6 show subimages of the two sites we used in our tests. A site consists of several images, generally aerial images of dimensions greater than 1000×1000 pixels. The two test sites are very different from each other: the first one is a mountainous rural area with several industrial facilities (Figure 5a), while the second is an urban area in flat terrain (Figure 6a). Figures 5b and 6b show curves used as seeds in the snake optimization process. Figures 5c and 6c show the results of the optimization. Although the building boundaries presented in Figures 6b and c appear very similar, careful inspection will reveal that there are significant differences between the two —

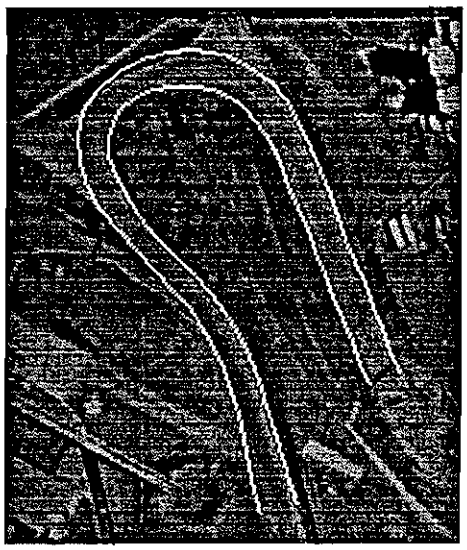
the optimized version is much more precise than the sketch. Finally, Figures 5d and 6d show the parameters provided by the system. The *Smoothness constraint* for the ribbon of the first site is relatively small because of the relatively high curvature of the ribbon. This aspect is captured by the context element *Seed min angle*. *Gaussian smoothing* needs to be smaller for the curve of the second site because of the relatively high edge density around the curve and, more particularly, the presence of shadow. This aspect is captured by the *Gradient mean* context element.



(a)



(b)

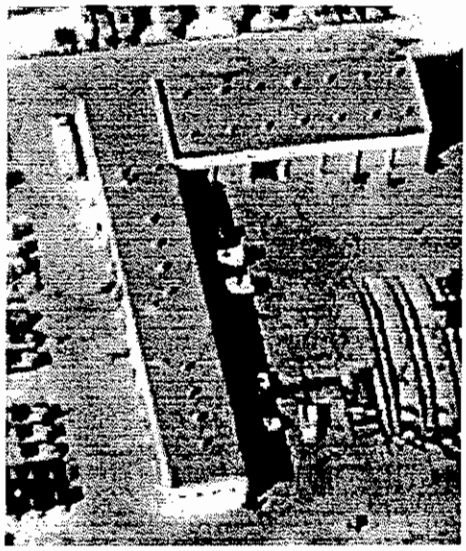


(c)

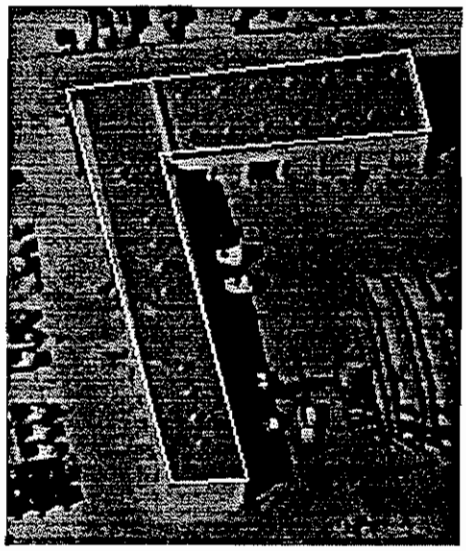
Parameters	Values
Type of snake	ribbon
Fixed endpoints	true
Gaussian smoothing	2
Initial step size	2.0
Stick length	10
Smoothness constraint	0.6
Width constraint	0.5
Curvature/tension ratio	1.0

(d)

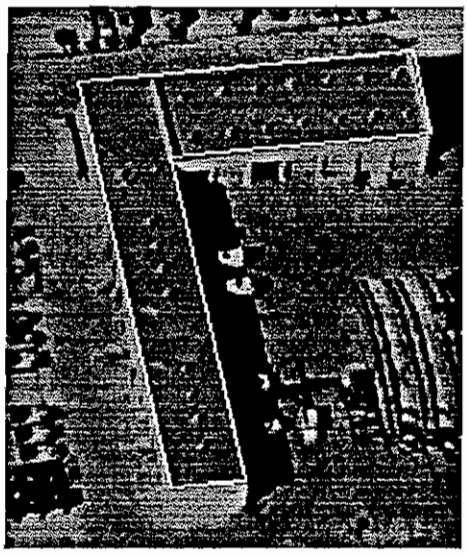
Figure 5: (a) Example of images of the first test site. (b) Ribbon seed curve. (c) Snake-optimized ribbon curve. (d) Provided parameters



(a)



(b)



(c)

Parameters	Values
Type of snake	Polygonal
Fixed endpoints	not used
Gaussian smoothing	1
Initial step size	2.0
Stick length	not used
Smoothness constraint	not used
Width constraint	not used
Curvature/tension ratio	not used

(d)

Figure 6: (a) Example of images of the second test site. (b) Closed 2–D curve. (c) Snake-optimized 2–D curve. (d) Provided parameters

9.2 System Evaluation

Testing the efficiency of our system poses three major problems with respect to maintaining test objectivity. The first one is the choice of the image curves to optimize. The snake algorithm requires some initial curves as input data. A bias in the experiments can be easily introduced by providing initial curves too close to the expected solution. In this case, the snake is going to converge toward the good solution whatever the parameters are.

The second problem is the order in which curves are optimized. Depending on the way evaluation is performed, another bias can be introduced by placing simpler examples at the end of the experiment, creating the illusion that the learning capability of the system has improved. Moreover, we have seen that the incremental scheme of retrieval is sensitive to the order of experiments. The best solution would be to define all the curves we want to test, and present them randomly to the user.

The last problem is the evaluation of the results. Some results may be acceptable for one user but not for another. This paper describes an ongoing study, and determining how well the learned result will carry over from one user to another has not yet been attempted. All the test results presented below come from a single user.

One very natural method of evaluation consists in using the system and counting the cumulative number of times the user has to adjust the result by hand. As the data base grows, the intervention required of the user should decrease, and so required adjustments should be fewer in number.

Results of this evaluation for the retrieval method based on the similarity measure expressed by Equation 2 are presented on Figure 7. We can see that for both sites, system reactivity is similar. There is a continuous decrease in the slope of each curve. This decrease is due to the effect of successfully learning suitable parameter assignments and represents an improvement in efficiency. The frequency of manual parameter setting that is required clearly decreases and tends toward zero, which is the theoretical ideal. The slope decrease also means that the user needs fewer trials to achieve his goal.

The third curve shows the hypothetical number of manual parameter settings for a user who does not use the learning module, but simply sets the parameters himself before each optimization. This curve fits the two others when the system starts to learn, and then tends to an asymptotic line with slope 2.0, indicating a mean of two manual settings per curve to optimize. The difference between

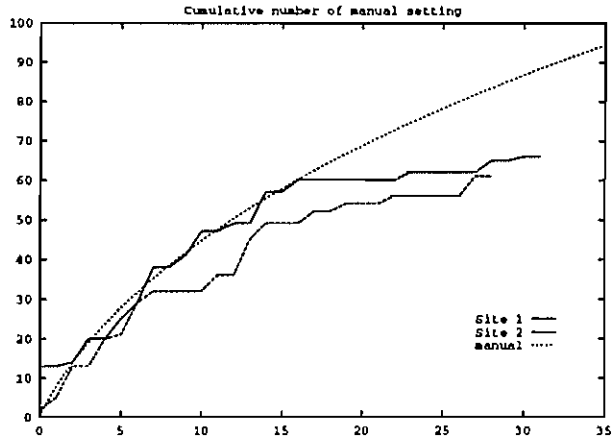


Figure 7: Cumulative number of manual settings of parameters

the amount of hypothetical manual parameter setting and the results obtained by a user employing the learning process indicates the improvement in efficiency provided by the learning module. From the graph it is apparent that the improvement increases as the system gains experience with the site.

Without the assistance of the parameter learning module, a novice user can require ten or more invocations of the snake algorithm before he attains a suitable parameter setting for each seed curve to be optimized. The capacity of the learning module to reduce the required number of invocations (to less than one per task in our experiments) represents a significant improvement in the efficiency with which snake algorithms can be employed in an interactive system.

9.3 Evaluation of Retrieval Procedure

This first experiment demonstrates clearly the learning ability of the system. However, it is sensitive to the three problems mentioned earlier: choice of original curves, order of examples, and evaluation of the results.

We therefore propose another method to evaluate more specifically the retrieval procedure. We consider the set of n curves that have already been optimized in the first experiment. For each curve $i \leq n$, we know $\mathbf{C_i}$ the contextual information associated with the curve, and $\mathbf{P_i}$ the parameter setting that has been used to find the optimized curve. Each context $\mathbf{C_i}$ is presented randomly to the

system that provides a set of n_1 possible parameters. Because we know what the correct parameter setting is ($\mathbf{P_i}$) we can measure the percentage of success of the system.

There are two ways to measure this percentage of success. The first one is to count the number of times parameter vector $\mathbf{P_i}$ belongs to the set of n_1 parameter vectors provided by the system. Because the system does not make constructive learning, and so does not propose new parameter values but instead only provides parameter vectors that have already been used, another measure is more accurate. It consists in counting the number of times parameter vector $\mathbf{P_i}$ belongs to the set of n_1 parameter vectors presented to the user when $\mathbf{P_i}$ was known to already belong to one of the categories of the system ($\mathbf{P_i}$ has already been used in a similar situation). Among the 68 optimized curves used in the experiments ($n = 68$), 39 appear to be in this situation[5].

This method is better than the first one for three reasons. First, randomly presenting the context vectors to the system is practically very easy to perform. Thus, we are going to be able to test the impact of example order on the different methods. Moreover, the user evaluation of the snake algorithm does not influence the measure that evaluates the performance of the system. Finally, the method is much less sensitive to the choice of the initial curves. Actually, this choice influences only the final structure of the category tree. "Too easy" curves tend to be associated with the same standard parameter setting, and thus always classified in the same category, thereby decreasing the number of categories and facilitating the categorization.

Experimental results:

Table 8 shows the performance results of the three different approaches based on conceptual clustering presented in Section 6: COBWEB, Anderson's method, and our method, which use the variance for numerical variables. COBWEB uses numerical context values as categorical ones. Anderson's and our method discriminate between numerical and categorical context elements. In all experiments, unless otherwise mentioned, the number n_0 of contexts considered as the nearest context in the category structure is set to 10, and n_1, the number of parameter settings presented to the user is set to 3.

Percentages presented in Table 8 are an average of success over twenty runs where examples are presented randomly to the system. As we can see, results are similar for all methods. This is explainable knowing that among the sixteen context elements used in this experiment, only two have

[5]Note that globally this number is independent of the order of the examples, even if the curves taken into account are not always the same.

	COBWEB	Anderson's	Variance method
site 1 + 2	77.6	78.9	78.1
site 1	83.9	83.3	81.7
site 2	85.8	86.5	83. 8

Table 8: Percentage of success of the different retrieval techniques.

numerical values that are different for each example (*gradient-mean* and *seed-mean-angle*). All other numerical values are associated with discrete functions or do not vary much (like *Sun angle*). These kinds of numerical values can be considered as categorical ones. Thus, the influence of the distinction between numerical and categorical values is low in this experiment. However, we can see that the system performs reasonably well, even with a small number of examples (39 examples on Site 1, 29 on Site 2).

Anderson's approach is favored by the method of parameter extraction we use after having selected the nearest contexts. Anderson's category structure is flat, and as a result, the number of context elements selected is generally greater than n_0. This is very sensitive when a small number of examples are present in the data base (as it is the case for Site 2). A priori knowledge required by the method has been set to some standard values. Other parameters required an exhaustive search among possible values to be set properly. Only one set of parameters gives correct categorization. Moreover, for the simple example presented in Section 6, the required setting was completely different, and very sensitive to the examples that were generated.

The tree structure produced by the hierarchical methods (COBWEB and Variance) are fairly similar for both methods. Analysis of normative context values[6] at the tree root shows that the classification first uses the *task* context element to categorize examples. Children of the root regroup examples with similar *task*. This is very satisfactory, because in the case of the snake algorithm, we know that the task is of first importance to determining a good parameter setting. For instance, roads and building delineation will require totally different settings.

Influence of example order:

As mentioned earlier, incremental schemes are sensitive to the order in which examples are presented. To test this influence, we compute the standard deviation over twenty runs of the number of successes for the three retrieval techniques. Results are presented in Table 9. Again, we can

[6]Context values V_{ij} that are present in class ω_k with a conditional probability $P(A_i = V_{ij}|\omega_k)$ greater than 0.67.

see that the values are similar. They are small, too. This means that the search structure of the categorization process does not affect the inference ability of the final category structure, as pointed out by Anderson [3].

	COBWEB	Anderson's	Variance method
site 1 + 2	1.52	1.54	1.43
site 1	0.79	0.80	0.80
site 2	0.58	0.85	0.78

Table 9: Standard deviation of the number of successes for the different retrieval techniques.

Influence of n_0 and n_1:

The only two parameters of the system are n_0, the number of contexts selected as being the final nearest contexts in the category structure, and n_1, the number of parameter settings presented to the user. Figure 8 shows the influence of n_1 (with $n_0 = 10$) on the percentage of success. As we can see, the percentage of success is over 50%, even with $n_1 = 1$. It means that more than half of the time, the first guess of the system was the good one. A value of 3 seems to be reasonable for n_1.

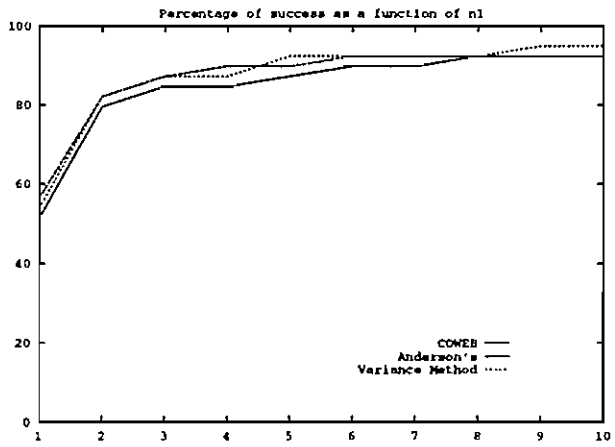


Figure 8: Influence of n_1 values over percentage of success.

As shown in Figure 9, the influence of n_0 is less important. A value $n_0 \geq 7$ is already sufficient to provide good results. It is important to note that the three methods presented use predictability to

categorize a new context and similarity to retrieve n_1 parameter settings from the n_0 nearest contexts from the new context. By increasing n_0, we give more importance to the similarity measure, and less to the predictability one. When n_0 equals n, the number of examples present in the data base, all three methods are equivalent to the one presented in Section 4 based only on similarity and with exhaustive search in the data base. As we can see in Figure 9, for $n_0 \geq 7$, performance of the variance method does not improve any more. It means that we have identical performances if the new context is categorized and then compared to only n_0 contexts, or if the new context is compared to the whole data base. This proves the ability of the heuristic search of categorization.

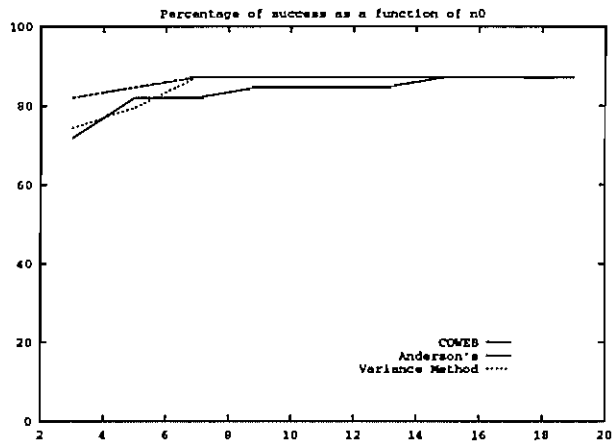


Figure 9: Influence of n_0 values over percentage of success.

Influence of number of context elements

We explained the similar performance of the different retrieval methods by the large number of categorical elements in the context vector we use. To draw a more accurate view of the retrieval methods, we performed tests on a subset of context elements containing three categorical, two numerical, and two ordinal context elements. These elements where chosen as the most important for categorization. Partial results we obtained are shown in Table 10. Performance of Anderson's and Variance method,s which separate categorical and numerical variables, are now better than COBWEB's method using only categorical variables.

Results of the three methods are also globally better than those shown in Figure 8. This points out an important factor : Context elements have to be adapted to the examples present in the data

	COBWEB	Anderson's	Variance method
site 1 + 2	82.2	83.8	85.4

Table 10: Percentage of success of the different retrieval techniques using a subset of seven context elements.

base. Our system has been designed for large data bases. The large number (sixteen) of context elements we use is not adapted to the small number of examples we have now in the data base. By removing context elements that are not essential because their values do not vary much in the whole data base, we improve the performance of the system. The automatic selection of context elements with regard to the number of examples already encountered is of primary interest in our future works.

Other future work remains on testing the ability of our system to choose an algorithm among several, and analyzing the various probabilities provided by the category structure. For instance, joint analysis of conditional probabilities $P(A_i = V_{ij}|\omega_k)$ and $P(\omega_k|A_i = V_{ij})$ should be an easy way to find out the discriminatory ability of each context element. This also should be a way to find a posteriori similarities between context element values. Computing this kind of similarity is essential for categorical context elements, and should greatly improve the search of similar contexts, and thus improve the retrieval performances.

10 Conclusion

This paper grew out of an attempt to solve a practical and important problem in interactive scene analysis: the automated selection of feature extraction algorithms and their parameters, as a function of image content and task requirements. An abstract characterization of this problem is that of mapping a multidimensional context space (representing image data and task specification) into a multidimensional algorithm-selection space (the extraction algorithms and their parameter settings). It was immediately apparent that an analytic design was infeasible. Two of the many reasons are

- We don't have effective ways of analytically describing image content.
- The range of possible tasks and image types is essentially infinite — no a priori design can hope to subsume all possible situations.

A "learning" approach appeared to be the only alternative.

The fact that the learning system is embedded in an interactive system (to deal with continuous change) offers both a challenge and an opportunity. The human operator must be aided rather than burdened by the presence of the learning system but can provide directed feedback about system performance.

Thus, the primary contribution of this paper lies in the structuring of an interactive, embedded learning system for an important problem in the design of computer vision systems — the automated selection of feature extraction algorithms and their parameters, as a function of image content, collateral data, and task requirements. The framework we have described lays the foundation for new learning mechanisms to be developed and tested — we have taken the first steps toward applying machine learning in a nonconventional learning context. We have also offered solutions to some of the subproblems that arise: how to define similarity of context vectors in which elements are both numerical and categorical, how to choose among the multiple parameter vectors that might be retrieved from the data base, and how to update the data base with experience gained through continual use of the feature extraction system. We have implemented and tested an initial design and demonstrated successful performance within a cartographic modeling domain using snake algorithms.

A Snakes

Here we provide a mathematically precise account of the snake algorithms that we have employed within our system for learning the parameters of vision algorithms.

A.1 2–D Linear Snakes

A 2–D snake is treated as a polygonal curve $\mathcal{C}$ defined by a set S containing n equidistant vertices

$$S = \{(x_i\ y_i),\ i = 1, \ldots, n\} \tag{A-1}$$

that can deform itself to maximize the average edge strength along the curve $\mathcal{G}(\mathcal{C})$:

$$\mathcal{G}(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \int_0^{|\mathcal{C}|} |\nabla \mathcal{I}(\mathbf{f}(s))|\ ds, \tag{A-2}$$

where I represents the image gray levels, s is the arc length of $\mathcal{C}$, $\mathbf{f}(s)$ is a vector function mapping the arc length s to points (x, y) in the image, and $|\mathcal{C}|$ is the length of $\mathcal{C}$. In practice, $\mathcal{G}(\mathcal{C})$ is computed by sampling the polygonal segments of the curve at regular intervals, looking up the gradient values $|\nabla \mathcal{I}(\mathbf{f}(s))|$ in precomputed gradient images, and summing them up. The gradient images are computed by gaussian smoothing the original image and taking the x and y derivatives to be finite differences of neighboring pixels. We have shown [12] that the points along a curve that maximizes $\mathcal{G}(\mathcal{C})$ are maxima of the gradient in the direction normal to the curve wherever the curvature of the curve is small. Therefore, such a curve approximates edges well except at corners. Unfortunately, $\mathcal{G}(\mathcal{C})$ is not convex functional and to perform the optimization, following Terzopoulos *et al.*, we minimize an energy $\mathcal{E}_(\mathcal{C})$ that is a weighted difference of a regularization term $\mathcal{E}_D(\mathcal{C})$ and of $\mathcal{G}(\mathcal{C})$:

$$\mathcal{E}_(\mathcal{C}) = \lambda_D \mathcal{E}_D(\mathcal{C}) - \lambda_G \mathcal{G}(\mathcal{C}) \tag{A-3}$$

$$\begin{aligned} \mathcal{E}_D(\mathcal{C}) &= \mu_1 \sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \\ &+ \mu_2 \sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2 \end{aligned} \tag{A-4}$$

The first term of $\mathcal{E}_D$ approximates the curve's tension and the second term approximates the sum of the square of the curvatures, assuming that the vertices are roughly equidistant. In addition, when starting, as we do, with regularly spaced vertices, this second term tends to maintain that regularity.

To perform the optimization we could use the steepest or conjugate gradient, but it would be slow for curves with large numbers of vertices. Instead, it has proven much more effective to embed the curve in a viscous medium and solve the equation of the dynamics

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial S} + \alpha \frac{dS}{dt} &= 0, \\ \text{with } \frac{\partial \mathcal{E}}{\partial S} &= \frac{\partial \mathcal{E}_D}{\partial S} - \frac{\partial \mathcal{G}}{\partial S}, \end{aligned} \tag{A-5}$$

where $\mathcal{E}$ is the energy of Equation A-3, α the viscosity of the medium, and S the state vector of Equation A-1 that defines the current position of the curve. Since the deformation energy $\mathcal{E}_D$ in Equation A-4 is quadratic, its derivative with respect to S is linear and therefore Equation A-5 can be rewritten as

$$\begin{aligned} K_S S_t + \alpha(S_t - S_{t-1}) &= -\left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \\ \Rightarrow (K_S + \alpha I) S_t &= \alpha S_{t-1} - \left.\frac{\partial \mathcal{E}}{\partial S}\right|_{S_{t-1}} \end{aligned} \tag{A-6}$$

where

$$\frac{\partial \mathcal{E}_D}{\partial S} = K_S S,$$

and K_S is a sparse matrix. Note that the derivatives of $\mathcal{E}_D$ with respect to x and y are decoupled so that we can rewrite Equation A-6 as a set of two differential equations in the two spatial coordinates

$$\begin{aligned} (K + \alpha I) X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I) Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \end{aligned}$$

where K is a pentadiagonal matrix, and X and Y are the vectors of the x and y vertex coordinates. Because K is pentadiagonal, the solution to this set of equations can be computed efficiently in $O(n)$ time using LU decomposition and backsubstitution. Note that the LU decomposition need be recomputed only when α changes.

In practice α is computed in the following manner. We start with an initial step size Δ_p, expressed in pixels, and use the following formula to compute the viscosity:

$$\alpha = \frac{\sqrt{2n}}{\Delta_p} \left| \frac{\partial \mathcal{E}}{\partial S} \right|, \tag{A-7}$$

where n is the number of vertices. This ensures that the initial displacement of each vertex is on the average of magnitude Δ_p. Because of the non linear term, we must verify that the energy has decreased from one iteration to the next. If, instead, the energy has increased, the curve is reset to its previous position, the step size is decreased, and the viscosity recomputed accordingly. This is repeated until the step size becomes less than some threshold value. In most cases, because of the presence of the linear term that propagates constraints along the whole curve in one iteration, it takes only a small number of iterations to optimize the initial curve.

The snakes described above have proved very effective at modeling smooth curves. Some objects, however, such as buildings, are best modeled as polygons with sharp corners. They can be handled in this context by completely turning off the smoothness term. Such objects typically have a relatively small number of corners, and the optimization is performed using a standard optimization technique.

A.2 3–D Linear Snakes

Snakes can be naturally extented to three dimensions by redefining C as a 3–D curve with n equidistant vertices $S = \{(x_i\ y_i\ z_i)\},\ i = 1,\ldots,n\}$ and considering its projections in a number of images for which we have accurate camera models. The average edge strength $\mathcal{G}(C)$ of Equation A-2 becomes the sum of the average edge strengths along the projection of the curve in the images under consideration, and the regularization term of Equation A-4 becomes

$$\begin{aligned} \mathcal{E}_D(C) &= \mu_1 \sum_i (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2 \\ &+ \mu_2 \sum_i (2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2 + (2z_i - z_{i-1} - z_{i+1})^2 \end{aligned} \tag{A-8}$$

Since the derivatives of $\mathcal{E}_D$ with respect to x, y, and z are still decoupled, we can rewrite Equation A-6 as a set of three differential equations in the three spatial coordinates:

$$\begin{aligned} (K + \alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \\ (K + \alpha I)Z_t &= \alpha Z_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Z}\right|_{Z_{t-1}} \end{aligned}$$

where X,Y, and Z are the vectors of the x,y, and z vertex coordinates.

The only major difference with the 2–D case is the use of the images' camera models. In practice, $\mathcal{G}(\mathcal{C})$ is computed by summing gradient values along the line segments linking the vertices' projections. These projections, and their derivatives, are computed from the state vector S using the camera models. Similarly, to compute the viscosity, we use the camera models to translate the average initial step Δ_p, a number of pixels, into a step Δ_w expressed in world units and use the latter in Equation A-7.

A.3 Ribbons

2–D snakes can also be extended to describe ribbon-like objects such as roads in aerial images. A ribbon snake is implemented as a polygonal curve forming the center of the road. Associated with each vertex i of this curve is a width w_i that defines the two curves that are the candidate road boundaries. The state vector S becomes the vector $S = \{(x_i\ y_i\ w_i)\},\ i = 1, \ldots, n\}$ and the average edge strength the sum of the edge strengths along the two boundary curves. Since the width of roads tends to vary gradually, we add an additional energy term of the form

$$\begin{aligned} \mathcal{E}_W(\mathcal{C}) &= \sum_i (w_i - w_{i-1})^2 \\ \Rightarrow \frac{\partial \mathcal{E}_W}{\partial W} &= LW, \end{aligned} \tag{A-9}$$

where W is the vector of the vertices' widths and L a tridiagonal matrix. The total energy can then be written as

$$\mathcal{E}_(\mathcal{C}) = \lambda_D \mathcal{E}_D(\mathcal{C}) + \lambda_W \mathcal{E}_W(\mathcal{C}) - \lambda_G \mathcal{G}(\mathcal{C})$$

and at each iteration the system must solve the three differential equations:

$$\begin{aligned} (K + \alpha I)X_t &= \alpha X_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial X}\right|_{X_{t-1}} \\ (K + \alpha I)Y_t &= \alpha Y_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial Y}\right|_{Y_{t-1}} \\ (K + \alpha I)W_t &= \alpha W_{t-1} + \left.\frac{\partial \mathcal{G}}{\partial W}\right|_{W_{t-1}} \end{aligned}$$

2–D ribbons can be turned into 3–D ones in exactly the same way 2–D snakes are turned into 3–D ones. The state vector S becomes the vector $S = \{(x_i\ y_i\ z_i\ w_i)\},\ i = 1, \ldots, n\}$ and at each iteration the system must solve four differential equations, one for each coordinate.

References

[1] W. Ahn and D.L. Medin. A two-stage model of category construction. *Cognitive Science*, 16:81,121, 1992.

[2] J.R. Anderson. *The adaptive character of thought.* Hillsdale, NJ: Erlbaum, 1990.

[3] J.R. Anderson. The adaptative nature of human categorization. *Psychological Review*, 18:409–429, 1991.

[4] V. Clement, G. Giraudon, S. Houzelle, and F. Sandakly. Interpretation of remotely sensed images in a context of multi sensor fusion using a multi-specialist architecture. *IEEE Trans. on Geoscience and Remote Sensing*, 1993.

[5] V. Clément and M. Thonnat. Integration of image processing procedures, ocapi : a knowledge-based approach. *CVGIP*, 57(2), March 1993.

[6] R.L. Cromwell and A.C. Kak. Automatic generation of object class descriptions using symbolic learning techniques. In *Proc. of the ninth National Conference on Artificial Intelligence*, pages 710,717, 1991.

[7] B.A. Draper, R.T. Collins, J. Brolio, A. R. Hanson, and E.M. Riseman. The schema system. *International Journal of Computer Vision*, 3(2):209–250, 1989.

[8] B.A. Draper, A. R. Hanson, and E.M. Riseman. Learning knowledge-directed visual strategies. In *DARPA Workshop on Image Understanding*, pages 933–940, 1992.

[9] B.A. Draper, A. R. Hanson, and E.M. Riseman. Statistical properties of learning recognition strategies. In *DARPA Workshop on Image Understanding*, pages 557,565, 1993.

[10] H.L. Dreyfus. La portee philosophique du connexionisme. In *Introduction aux sciences cognitives.* In French, Editions Folio, 1987.

[11] D.H. Fisher. *Knowledge Acquisition Via Incremental Conceptual Clustering*, pages 139,172. Kluwer Academic Publishers, Boston, 1987.

[12] P. Fua and Y.G. Leclerc. Model driven edge detection. *Machine Vision and Applications*, 3:45–56, 1990.

[13] A. Giacometti. *Modeles hybrides de l'expertise.* PhD dissertation, Telecom Paris. In French, 1993.

[14] A. J. Hanson and L. Quam. Overview of the sri cartographic modeling environment. In *DARPA Workshop on Image Understanding*, pages 576–582, April 1988.

[15] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[16] Y. Kodratoff and G. Tecuci. Learning based on conceptual distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 10(6):897,909, November 1988.

[17] P. Langley, J.M. Zytkow, H.A. Simon, and G.L. Bradshaw. *The search for Regularity: Four Aspects of Scientific Discovery*, pages 425,469. Morgan Kaufmann Publishers Inc., Los Altos, CA, 1986.

[18] M. Lebowitz. *Concept Learning in a Rich Input Domain*, pages 193–214. Morgan Kaufmann Publishers Inc., Los Altos, CA, 1986.

[19] M. Lebowitz. *Experiments with Incremental Concept Formation: UNIMEM*, pages 103,138. Kluwer Academic Publishers, Boston, 1987.

[20] D. M. McKeown, W. A. Harvey, and J. McDermott. Rule-based interpretation of aerial imagery. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 7(5):570–585, September 1985.

[21] R.S. Michalski. Pattern recognition as rule-guided inductive inference. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-2(4):349,361, July 1980.

[22] R.S. Michalski and R.E. Stepp. Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-5(4):396,409, July 1983.

[23] J.L. Mundy, R. Welty, L. Quam, T. Strat, W. Bremmer, M. Horwedel, D. Hackett, and A. Hoogs. The radius common development environment. In *Proc. of AIPR, Washington, DC*, October 1991. Also *in Proc. of DARPA Image Understanding Workshop*, San Diego, California, 1992.

[24] R.E. Stepp and R.S. Michalski. *Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects*, pages 471,498. Morgan Kaufmann Publishers Inc., Los Altos, CA, 1986.

[25] T. M. Strat and M. A. Fischler. Context-based vision: Recognizing objects using both 2d and 3d imagery. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(10):1050–1065, October 1991.

[26] Thomas M. Strat. *Natural Object Recognition.* Springer-Verlag, New York, 1992.

[27] Thomas M. Strat. Employing contextual information in computer vision. In *DARPA Workshop on Image Understanding*, 1993.

[28] D. Terzopoulos, A. Witkin, and M. Kass. Symmetry-seeking models and 3D object reconstruction. *International Journal of Computer Vision*, 1:211–221, 1987.

[29] Z.W. Wang, S.K.M. Wong, and Y.Y. Yao. An analysis of vector space models based on computational geometry. In *ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 152–160, June 1992.

[30] S. Wrobel. *Demand-driven Concept Formation*, pages 289,319. Springer Verlag, 1989.