A Lattice Machine Approach to Automated Casebase Design: Marrying Lazy and Eager Learning

Hui Wang, Werner Dubitzky, Ivo Düntsch, David Bell

School of Information and Software Engineering, University of Ulster

Newtownabbey, BT 37 0QB, N.Ireland

{H.Wang,W.Dubitzky,I.Duentsch,DA.Bell}@ulst.ac.uk

Abstract

Case-based reasoning (CBR) is concerned with solving new problems by adapting solutions that worked for similar problems in the past. Years of experience in building and fielding CBR systems have shown that the "case approach" is not free from problems. It has been realized that the knowledge engineering effort required for designing many real-world casebases can be prohibitively high. Based on the wide-spread use of databases and powerful machine learning methods, some CBR researchers have been investigating the possibility of designing casebases automatically. This paper proposes a flexible model for the automatic discovery of abstract cases from databases based on the Lattice Machine. It also proposes an efficient and effective algorithm for retrieving such cases. Besides the known benefits associated with abstract cases, the main advantages of this approach are that the discovery process is fully automated (no knowledge engineering costs).

Keywords: case-based reasoning, machine learning, knowledge acquisition, automated modeling

1 Introduction

Case-based reasoning (CBR) is concerned with solving new problems by adapting solutions that worked for similar problems in the past. CBR research is motivated by the desire to establish cognitive models to understand human thinking and behavior (psychology, cognitive science), and to build more effective, efficient, and robust computer systems that solve real-world problems (artificial intelligence). Since the early nineties, CBR has been successfully applied in a wide variety of areas. However, with almost ten years of both theoretical and applied experience in building and fielding case-based systems, it has been realized that the "case approach" is not free from problems [Leake, 1996]. Two of the more important issues that need to be addressed are outlined below:

- the implementation of *casebase maintenance* policies for revising the organization of casebases in order to facilitate effective and efficient future reasoning [Leake and Wilson, 1998]; and
- the *automation* of the *case engineering* process, that is, the automatic generation of case-knowledge components from existing information (i.e., databases) [Patterson *et al.*, 1998]; the work presented in this paper will focus on this issue.

The term *case engineering* refers to task of designing a casebase, that is, the processes of generating those components that represent the application-specific knowledge contained in a CBR system. In general, such knowledge structures describe the content structure of cases, criteria for organizing cases within and retrieving cases from the case library (indexing scheme, similarity measures), rules for adapting case solutions, and case revision and retention schemes [Lenz *et al.*, 1998]. In addition to determining the fundamental knowledge elements, the generation process also entails the selection or extraction of those cases that will be used to populate the initial casebase.

The work presented in this paper proposes *Lattice Machine* – a formal framework for learning from relations, and its application to automatically construct (case content description) and extract (case selection) abstract cases from databases. Furthermore, an efficient algorithm for the retrieval of such abstract cases is also proposed.

The remainder of the paper is organized as follows. Section 2 briefly discusses the role of abstract cases in CBR. Section 3 introduces the definitions and notational conventions, followed in Section 4 by a formal description of Lattice Machine, the pivotal part of the work in this paper. Sections 5 and 6 present efficient algorithms for extracting abstract cases and retrieving cases, based on the theoretical results in Lattice Machine. Experimental results are reported in Section 7. Section 8 concludes the paper.

2 Abstract Cases in CBR

Traditional CBR systems retrieve, reuse, and retain cases in a representation reflecting concrete problem-solving episodes. Recently, researchers have investigated

the role and use of abstract cases, e.g., [Bergmann and Wilke, 1996]. Abstract cases represent cases at a higher level of abstraction. Through abstract cases, the CBR process can be supported in several ways [Bergmann and Wilke, 1996]; those ways pertinent to the discussion in this paper are outlined below:

- Abstract cases can reduce the complexity of a casebase by substituting sets of concrete cases and thereby significantly reducing the size of the casebase. A drastically reduced casebase can improve retrieval efficiency, reduce maintenance costs, and eliminate or alleviate the notorious swamping problem [Smyth and Keane, 1995].
- Cases at higher levels of abstraction can serve as prototypes for indexing larger sets of more detailed cases. This can have profound effects on reducing retrieval times, maintenance costs, and it can promote a better user understanding of the casebase content, and facilitate explanations for the system's reasoning process.

In most situations, abstract cases are not readily available, they must be generated – manually or automatically – from concrete cases. To manually construct abstract cases will require a very high knowledge engineering effort for most applications. Whereas an automatic generation procedure requires general domain knowledge about ways of mapping concrete cases onto higher levels of abstraction.

The abstract case construct proposed in this paper is based on the concept of hypertuples. Hypertuples, i.e., abstract cases, are automatically generated through a so-called domain lattice, which is implied in a problem domain. The extraction and retrieval of abstract cases are achieved in domain lattice using the Lattice Machine.

3 Definitions and notation

To present our findings concisely and within the given page limit, we briefly introduce some notational conventions and definitions that are used throughout the paper.

3.1 Decision systems

An *information system* is a tuple $\mathcal{I} = \langle U, \Omega, V_x \rangle_{x\in\Omega}$, where $U = \{a_0, \ldots, a_N\}$ is a nonempty finite set and $\Omega = \{x_0, \ldots, x_T\}$ is a nonempty finite set of mappings $x_i : U \to V_{x_i}$ [1].

We interpret U as a set of objects and Ω as a set of attributes or features, each of which assigns to an object a its value under the respective attribute. Let $\mathbf{V} \stackrel{\text{def}}{=} \prod_{x\in\Omega} V_x$. For $a \in U$, we let $\Omega(a) \stackrel{\text{def}}{=} \langle x(a) \rangle_{x\in\Omega} \in \mathbf{V}$. Each $\Omega(a)$ is called a *tuple*, and the collection of all tuples is denoted by $\mathbf{D}$. Thus, for each $t \in \mathbf{D}$, there is at least one $a \in U$ such that $\Omega(a) = t$.

[1] Note V_{x_i} can be finite or infinite. For the latter case the domain lattice is infinite. However we are only interested in the *finite* sublattice generated from a finite casebase.

A *decision system* $\mathcal{D}$ is a pair $\langle \mathcal{I}, d \rangle$, where $\mathcal{I}$ is an information system as above, and $d : \mathbf{D} \twoheadrightarrow V_d = \{d_0, \ldots, d_{\mathbf{K}}\}$ is an onto mapping, called a *labeling* of $\mathbf{D}$; the value $d(t)$ is called the *label of* t.

The mapping d induces a partition $\mathcal{P}_d$ of $\mathbf{D}$ with the classes $\{\mathbf{D}_0, \ldots, \mathbf{D}_{\mathbf{K}}\}$, where $t \in \mathbf{D}_i \iff d(t) = d_i$.

In this paper we consider a dataset represented as a decision system $\mathcal{D}$, which can be regarded as an *initial casebase* consisting of concrete cases. Then $\mathbf{D}$ is the set of (descriptions of) concrete cases, d is the case solution, and $\mathbf{V}$ is the set of all possible concrete cases in a problem domain. Therefore each $t \in \mathbf{D}$ is associated with a solution in the form of a class label $d(t)$.

3.2 Order and lattices

Let $\mathcal{P} = \langle P, \leq \rangle$ be a partially ordered set and $T \subseteq P$. We let $\downarrow T \stackrel{\text{def}}{=} \{y \in P : (\exists x \in T)\ y \leq x\}$. If no confusion can arise, we shall identify singleton sets with the element they contain.

Let $\mathcal{L}$ be a lattice, partially ordered by $\leq$. For $x, y \in \mathcal{L}$, the least upper bound (or sum) is written by $x + y$ and the greatest lower bound (or product) by $x \times y$. For $A \subseteq \mathcal{L}$, its least upper bound and greatest lower bound are denoted by $\text{lub}(A)$ and $\text{glb}(A)$ respectively. An element $a \in A$ is called *maximal* in A, if for all $x \in A$, $a \leq x$ implies $x = a$.

For $A, B \subseteq \mathcal{L}$, we say that B *covers* A or A *is covered by* B, written as $A \preccurlyeq B$ if for each $s \in A$ there is some $t \in B$ such that $s \leq t$.

The *sublattice* of $\mathcal{L}$ generated from $M \subseteq \mathcal{L}$, written by $[M]$, is $[M] = \{t \in \mathcal{L} : \exists X \subseteq M \text{ such that } t = \text{lub}(X)\}$. The greatest element in $[M]$ is $\text{lub}(M)$. If M is finite, $[M]$ is also finite.

A comprehensive discussion on lattice theory can be found in [Grätzer, 1978].

4 The Lattice Machine

This section introduces the Lattice Machine, a construct which facilitates the discovery of abstract cases from a given dataset. The discussion should also make apparent how the Lattice Machine is linked to machine learning concepts.

4.1 Domain lattice

We have found that, given a dataset expressed as a decision system, an elegant mathematical structure (lattice) is implied. This structure makes it possible to investigate CBR, machine learning, as well the relationship between the two from an algebraic perspective. In the sequel, we shall use $\mathcal{D}$ as described above as a generic decision system representing a dataset.

Let $\mathcal{L} \stackrel{\text{def}}{=} \prod_{x\in\Omega} 2^{V_x}$. Then $t \in \mathcal{L}$ is a vector $\langle t(x) \rangle_{x\in\Omega}$, where $t(x) \subseteq V_x$ are sets of values [2]. The elements of $\mathcal{L}$ are called *hypertuples*; the elements t of $\mathcal{L}$ with $|t(x)| = 1$ for all $x \in \Omega$ are called *simple tuples*. Any set of

[2] Note that if $t \in \mathcal{L}$ and $x \in \Omega$, then $t(x)$ is the projection of t to its x–th component.

hypertuples is called a *hyperrelation* [3]. Note that $\mathbf{V}$ is a set of *all* simple tuples for a given problem domain, and $\mathbf{D}$ is the set of simple tuples described in the dataset $\mathcal{D}$.

$\mathcal{L}$ is a lattice under the ordering

$$t \leq s \iff t(x) \subseteq s(x) \tag{1}$$

with the sum and product operations, and the maximal element (i.e., 1) given by

$$t + s = \langle t(x) \cup s(x) \rangle_{x \in \Omega}. \tag{2}$$

$$t \times s = \langle t(x) \cap s(x) \rangle_{x \in \Omega}, \tag{3}$$

$$1 = \langle V_x \rangle_{x \in \Omega}. \tag{4}$$

$\mathcal{L}$ is called *domain lattice* for $\mathcal{D}$.

There is a natural embedding of $\mathbf{D}$ into $\mathcal{L}$ by assigning

$$\Omega(a) \mapsto \langle \{x_0(a)\}, \{x_1(a)\}, \ldots, \{x_T(a)\} \rangle.$$

and we shall identify $\mathbf{D}$ with the image of this embedding. Then $\mathbf{D} \subseteq \mathbf{V} \subseteq \mathcal{L}$.

In the context of CBR, simple tuples are concrete cases whereas hypertuples are abstract cases since hypertuples cover multiple simple tuples hence they are "abstractions" of simple tuples.

Table 1(a) is a dataset (decision system) consisting of three simple tuples, where $V_{X_1} = \{a, b\}$ and $V_{X_2} = \{0, 1\}$, and d is the labeling. Table 1(b) and (c) are sets of hypertuples, which are the least and greatest E-sets respectively for the dataset, to be defined later.

$$\begin{aligned}
\mathbf{D} &= \{\langle a, 0\rangle, \langle a, 1\rangle, \langle b, 0\rangle\};\\
\mathbf{V} &= \{\langle a, 0\rangle, \langle a, 1\rangle, \langle b, 0\rangle, \langle b, 1\rangle\};\\
\mathcal{L} &= \{\langle \emptyset, \emptyset\rangle, \langle \emptyset, \{0\}\rangle, \langle \emptyset, \{1\}\rangle, \langle \emptyset, \{0, 1\}\rangle,\\
&\quad \langle \{a\}, \emptyset\rangle, \langle \{a\}, \{0\}\rangle, \langle \{a\}, \{1\}\rangle, \langle \{a\}, \{0, 1\}\rangle,\\
&\quad \langle \{b\}, \emptyset\rangle, \langle \{b\}, \{0\}\rangle, \langle \{b\}, \{1\}\rangle, \langle \{b\}, \{0, 1\}\rangle,\\
&\quad \langle \{a, b\}, \emptyset\rangle, \langle \{a, b\}, \{0\}\rangle, \langle \{a, b\}, \{1\}\rangle, \langle \{a, b\}, \{0, 1\}\rangle\}.
\end{aligned}$$

U	X_1	X_2	d
u_0	a	0	α
u_1	a	1	α
u_2	b	0	β

(a)

U	2^{X_1}	2^{X_2}	d
u'_0	$\{a\}$	$\{0, 1\}$	α
u'_1	$\{b\}$	$\{0\}$	β

(b)

U	2^{X_1}	2^{X_2}	d
u'_0	$\{a\}$	V_{X_2}	α
u'_1	$\{b\}$	V_{X_2}	β

(c)

Table 1: *(a) A set of simple tuples in a decision system. (b) A set of hypertuples as the least E-set. (c) A set of hypertuples as the greatest E-set.*

[3]The concept of hyperrelation has been used before in e.g. [Orlowska, 1985; Wang *et al.*, 1998].

4.2 Equilabelledness and generalization

A dataset imposes a labeling d of $\mathbf{D}$ on the domain lattice $\mathcal{L}$. Thus all elements in $\mathbf{D}$ are *labeled*, and those in $\mathbf{V} \setminus \mathbf{D}$ are *unlabeled*. This labeling can be generalized to elements in the lattice which cover $\mathbf{D}$. This generalization must be consistent with d in the sense that the generalized labeling must be the same as d for $t \in \mathbf{D}$. This renders only those generalizations acceptable which generalize d to *equilabeled* elements. Intuitively, an equilabeled element is $t \in \mathcal{L}$ which covers at least one labeled element and all labeled elements covered by t have the same label.

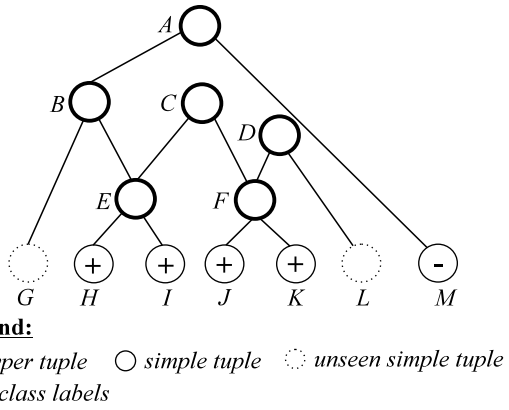

Figure 1: A running example.

For an illustrative example, consider the diagram in Figure 1, which depicts a small part of a domain lattice. The bottom elements (thin-lined circles) in the diagram represent simple tuples. All other elements (bold-lined circles) represent hypertuples. The original labeling d is defined only for the elements $\{H, I, J, K, M\}$. Here, B is equilabeled as H and I are all labeled positive by d while G is unlabelled. In fact, all elements in Figure 1 are equilabeled except A, G and L: A covers elements with different labels whereas G and L cover no labeled element.

Formally, we call an element $r \in \mathcal{L}$ *equilabeled* with respect to $\mathbf{D}_q$, if $\emptyset \neq\downarrow r \cap \mathbf{D}$, $\downarrow r \cap \mathbf{D} \subseteq \mathbf{D}_q$. In other words, r is equilabeled if $\downarrow r$ intersects $\mathbf{D}$, and every element in this intersection is labeled d_q for some $q \leq \mathbf{K}$. Recall that $\mathbf{K}$ is the number of classes. In this case, we say that r *G-belongs to* $\mathbf{D}_q$. We denote the set of all equilabeled elements G-belonging to $\mathbf{D}_q$ by $\mathcal{E}_q$, and let $\mathcal{E}$ be the set of all equilabeled elements. Note that $\mathbf{D} \subseteq \mathcal{E}$, and that $q, r \leq \mathbf{K}, q \neq r$ implies $\mathcal{E}_q \cap \mathcal{E}_r = \emptyset$.

We will now extend d over all of $\mathcal{L}$ by setting

$$d(r) = \begin{cases} d_q, & \text{if } r \in \mathcal{E}_q, \\ unknown, & \text{otherwise.} \end{cases}$$

Now $\mathcal{E}$, along with the extended labeling, can be regarded as a casebase of abstract cases (hypertuples). This is clearly too large. Since the elements in $\mathcal{E}$ are partially ordered – some are covered by some others –

we need only look at those which are not covered by any; they are *maximal*. Our wish to find maximal elements in some context leads to the following notions.

Def. 4.1. A *(generalization) context* (for learning) is a set P such that $\mathbf{D} \subseteq P \subseteq \mathbf{V}$. We let

$$\mathbf{M}(P) \stackrel{\text{def}}{=} \{h \in \mathcal{E} : \exists X \subseteq P, h = \text{lub}(X)\}$$

$$\mathbf{E}(P) \stackrel{\text{def}}{=} \{t : t \text{ is maximal in } \mathbf{M}(P)\}.$$

$\mathbf{E}(P)$ is called the *E–set for* P, and $t \in \mathcal{L}$ is said to be *in context* P if $t \in \mathbf{M}(P)$.

We observe the following lemma [4]:

Lemma 4.2. *$A \subseteq B \subseteq \mathcal{L}$ implies $\mathbf{M}(A) \preccurlyeq \mathbf{M}(B)$ and $\mathbf{E}(A) \preccurlyeq \mathbf{E}(B)$.*

This lemma implies $\mathbf{E}(\mathbf{D}) \preccurlyeq \mathbf{E}(P) \preccurlyeq \mathbf{E}(\mathbf{V})$ for $\mathbf{D} \subseteq P \subseteq \mathbf{V}$. We then call $\mathbf{E}(\mathbf{D})$ *least E-set* and $\mathbf{E}(\mathbf{V})$ *greatest E-set*, which are denoted by $\mathbf{E}$ and $\mathbb{E}$ respectively. It is not hard to see that $\mathbb{E}$ is the set of all maximal elements in $\mathcal{E}$.

Consider Figure 1 again. $\mathbf{E}(\{H, \cdots, K, M\}) = \{C, M\}$, and $\mathbf{E}(\{G, \cdots, M\}) = \{B, C, D, M\}$.

4.3 Interpretation of a domain lattice

Given a hypertuple t in a context P (i.e., $t \in \mathbf{M}(P)$), we need a calculus to get the remaining hypertuples in the same context (see Theorem 4.1 below). We therefore need to interpret the domain lattice in a suitable way.

Formally, let $t = \langle e_0, \ldots, e_T \rangle$ be a hypertuple, with $e_i = \{e_{i_0}, \ldots, e_{i_{t(i)}}\}$. We think of t as an (undeterministic) description of some object a in the following way:

$$(x_0(a) = e_{0_0} \vee \ldots \vee x_0(a) = e_{0_{t(0)}}) \wedge \ldots$$
$$\ldots \wedge (x_T(a) = e_{T_0} \vee \ldots \vee x_T(a) = e_{T_{t(T)}}).$$

In short we have $t = e_0 \wedge \cdots \wedge e_T$.

Each e_i stands as a hypertuple on its own, which is interpreted as $\langle V_0, \cdots, V_{i-1}, e_i, V_{i+1}, \cdots, V_T \rangle$; and $e_j \wedge e_k$ is interpreted as $\text{glb}(e_j, e_k)$.

Consider a hypertuple $t = \langle e_0, \cdots, e_T \rangle$. The *relative complement* or, simply, *R-complement* of t with respect to the greatest element in $\mathcal{L}$ (i.e., 1) is $\bar{t} = \overline{e_0} \vee \cdots \vee \overline{e_T}$, and $\overline{e_j}$ is interpreted as $V_j \setminus e_j$.

Given a hyperrelation $R = \{t_0, \cdots, t_n\}$, where $t_i = \langle e_{i0}, \cdots, e_{iT} \rangle$, we wish to calculate its R-complement $\overline{R}$ in such a way that $R \cup \overline{R} \succcurlyeq \mathbf{V}$ and $\downarrow R \cap \downarrow \overline{R} = \emptyset$. For this purpose we interpret R as $t_0 \vee \cdots \vee t_n$. Using the propositional calculus we can calculate the R-complement of R as follows. Let $X = \{\bigwedge_{i=0}^{n} \overline{e_{ij_i}} : \text{ for } j_i \in \{0, \cdots, T\}\}$. $\overline{R} = \bigwedge_{i=0}^{n} \overline{t_i} = \bigvee_{x \in X} x$. If follows $|X| = (T+1)^{(n+1)}$. Clearly the calculation of R-complement is exponential, which makes this not practical for large R.

The above R-complement calculus can be generalized as follows. Let $t = \langle e_{t0}, \cdots, e_{tT} \rangle$ and $h = \langle e_{h0}, \cdots, e_{hT} \rangle$. If $t \leq h$ then the R-complement of t with respect to h, denoted by $\overline{t|h}$, is calculated similarly except that $\overline{e_{tj}} = e_{hj} \setminus e_{tj}$.

[4] Due to lack of space, we omit all proofs throughout the paper.

Lemma 4.3. *For $R \subseteq \mathcal{L}$,*

1. *$R \cup \overline{R} \succcurlyeq \mathbf{V}$.*
2. *$\downarrow R \cap \downarrow \overline{R} = \emptyset$.*
3. *Let $W \stackrel{\text{def}}{=} \{w \in \mathcal{L} : \downarrow w \cap \downarrow R = \emptyset\}$. Then $\overline{R}$ is the set of all maximal elements in W.*

The first two properties guarantee that $\overline{R}$ is exclusively complementary to R with respect to the maximal element of the domain lattice. For an example, consider Table 1(a). The table is interpreted as $u_0 \vee u_1 \vee u_2$. u_0 is interpreted as $(X_1 = a) \wedge (X_2 = 0)$. $\overline{u_0} = \overline{\langle a, 0 \rangle} = \bar{a} \vee \bar{0} = \langle \bar{a}, V_2 \rangle \vee \langle V_1, \bar{0} \rangle$. Let $V_1 = \{a, b\}$ and $V_2 = \{0, 1\}$. Then $\overline{u_0} = \langle b, \{0, 1\} \rangle \vee \langle \{a, b\}, 1 \rangle$, which clearly covers u_1 and u_2, as well as an unseen tuple $\langle b, 1 \rangle$.

The third property says that $\overline{R}$ is the set of maximal elements not covered by R. Consider Figure 1. Let $R = \{E\}$. Then $\overline{R} = \{G, D, M\}$.

4.4 Hypothesis space and casebase

Given a dataset we wish to have a *hypothesis* to replace the entire dataset. The hypothesis should not only cover the dataset but also generalize it. As discussed earlier, a dataset $\mathcal{D}$ imposes a labeling on the underlying domain lattice. The labeling can then be generalized to elements in $\mathbf{M}(P)$ for a given context P. However we do not need to use the whole $\mathbf{M}(P)$ as the hypothesis; a proper subset of it will suffice. Then a hypothesis is just a set of hypertuples, each of which is more informative than the simple tuples in the original dataset. Therefore, it is possible to consider a *casebase* as a hypothesis for a dataset. In this section we will introduce and justify some concepts through which we can precisely describe what kind of hypothesis we are aiming for.

Note that, by Lemma 4.2, $\mathbf{M}(\mathbf{D}) \preccurlyeq \mathbf{M}(P) \preccurlyeq \mathbf{M}(\mathbf{V})$ for $\mathbf{D} \subseteq P \subseteq \mathbf{V}$. Therefore $H \subseteq \mathbf{M}(P) \Rightarrow H \subseteq \mathbf{M}(\mathbf{V})$. Then we have

Def. 4.4. A *hypothesis for* $\mathbf{D}$ is a $H \subseteq \mathbf{M}(\mathbf{V})$ such that $\mathbf{D} \preccurlyeq H$. We use $\text{GEN}(\mathbf{D})$ to denote the set of all hypotheses for $\mathcal{D}$.

Similarly, we define a *hypothesis for* $\mathbf{D}_q$. Note that for a hypothesis H for $\mathbf{D}$, $H \cap \mathcal{E}_q$ is a hypothesis for $\mathbf{D}_q$. Conversely, if H_q is a hypothesis for $\mathbf{D}_q$ for each $q \leq \mathbf{K}$, then $H \stackrel{\text{def}}{=} \bigcup_{q \leq \mathbf{K}} H_q$ is a hypothesis for $\mathbf{D}$.

Since $\mathbf{M}(P)$ contains only equilabeled elements, H is *consistent* with the dataset. Since $\mathbf{D} \preccurlyeq H$, H covers all simple tuples in the dataset.

Def. 4.5. Let H_j and H_k be two hypotheses for $\mathbf{D}$. Then H_j is **more general than** H_k if and only if $H_k \preccurlyeq H_j$. H_j is **(strictly) more general than** H_k, written $(H_k \prec H_j)$, if and only if $(H_k \preccurlyeq H_j) \wedge (H_j \not\preccurlyeq H_k)$.

A hypothesis H for $\mathbf{D}$ is *maximally general* if and only if $H \in \text{GEN}(\mathbf{D})$ and there is no $H' \in \text{GEN}(\mathbf{D})$ such that $H \prec H'$. We denote by $\mathbf{G}$ the set of all maximally general hypotheses for $\mathbf{D}$. In [Mitchell, 1997] $\mathbf{G}$ is called the *general boundary* for $\mathbf{D}$.

The following lemma establishes the equivalence between $\mathbf{G}$ and the greatest E-set ($\mathbb{E}$).

Lemma 4.6. $\mathbf{G} = \{\mathbb{E}\}$.

This lemma says that, although there are many possible hypotheses for a given dataset, there is only one maximally general hypothesis – the greatest E-set. This hypothesis is consistent with the dataset but has the maximal coverage of unseen simple tuples (because it has the maximal context $\mathbf{V}$). General boundary is a well established concept in the field of concept learning, and it has been used as an inductive bias in some concept learning algorithms [Mitchell, 1997]. The equivalence of the greatest E-set and the general boundary enables us to use the same inductive bias in automatic casebase design. Therefore our objective is to find the greatest E-set for a dataset. Our casebase design and retrieval are both associated with the greatest E-set.

However, as shown in [Haussler, 1988], the size of the general boundary can grow exponentially in the number of training examples. In the context of domain lattice, calculating the greatest E-set needs $\mathbf{V}$, which is not explicitly available; instead it has to be calculated from $\mathbf{D}$ using the R-complement calculus discussed above. This involves calculating $\overline{\mathbf{D}}$, which has been shown exponential in $|\mathbf{D}|$. It is then not practical to directly use the greatest E-set as the casebase. The following theorem guarantees that we can use a much smaller hypothesis as the casebase, but we can still use the general boundary as inductive bias. In effect it establishes the relationship between the casebase and the greatest E-set.

Theorem 4.1. *Let H be hypothesis for $\mathcal{D}$, as defined in Def. 4.4. Let $\mathbf{V}_{\mathbb{E}} = \downarrow \mathbb{E} \cap \mathbf{V}$, and $\mathbf{V}_H = \{t \in \mathbf{V} : \exists h \in H, \exists g \in \overline{(h|h+t)} \cup h$, such that $t \leq g$, and g is equilabeled or unknown$\}$. Then $\mathbf{V}_{\mathbb{E}} \subseteq \mathbf{V}_H$.*

This theorem says that if a case (simple tuple) t is covered by $\mathbb{E}$, then there must be h such that $t \leq h$ or $t \leq g$ for $g \in \overline{h|h+t}$ with g being either equilabeled or unknown. Note that $\overline{h|h+t}$ is the R-complement of h with respect to $h+t$, which is the set of all elements covered by $h+t$ but not covered by h. The CaseRetrieve algorithm in Section 6 exploits this theorem to retrieve cases to classify new cases.

Theorem 4.1 says that any hypothesis can be used as a casebase that serves as an intermediary between a new case and the greatest E-set. Classification of a new case can then be made based on its relationship to the greatest E-set, which employs the general boundary inductive bias.

5 Case extraction

As indicated in Theorem 4.1, an expected casebase can be any hypothesis as defined in Def. 4.4. The simplest one is the dataset itself. However, checking whether the conditions are satisfied requires computing R-complements of the tuples. It is usually the case that datasets are large hence the computation cost is high. Therefore we need an algorithm to efficiently find a hypothesis, other than the dataset itself, satisfying the conditions. The least E-set seems ideal since it is the E-set in the minimal context ($\mathbf{D}$). However calculating the least E-set is computationally expensive. The following algorithm, CaseExtract, finds, given the minimal context $\mathbf{D}$, the set of elements in $\mathbf{M}(\mathbf{D})$ which have disjoint coverage of $\mathbf{D}$.

Given $\mathbf{D}_q$ and $\mathcal{E}_q$ as defined above.

- Initialization: let $X = \mathbf{D}_q, H = \emptyset$.
- Repeat until X is empty:
  1. Let $h \in X$ and $X = X \setminus \{h\}$.
  2. For $g \in X$, let $X = X \setminus g$. If $h+g$ is equilabeled then $h = h + g$.
  3. Let $H = H \cup \{h\}$.

This algorithm bi-partitions X into a set of elements the sum of which is an equilabeled element, and a new X consisting of the rest of the elements. The new X is similarly bi-partitioned until X becomes empty. This process leads to a binary tree, the depth of which is a measure of the time complexity of the algorithm. In the worst case the time complexity for building the casebase for class $\mathbf{D}_q$ is in the order of $O(|\mathbf{D}_q|)$. Therefore the worst case complexity for building the whole casebase is $O(\mathbf{K} \times |\mathbf{D}_q|)$, where $\mathbf{K}$ is the number of classes.

Consider Figure 1. CaseExtract gives the hypothesis $\{E, F, M\}$ which has disjoint coverage of the labeled elements.

6 Case retrieval

The retrieval of relevant cases from the casebase is arguably the most important process in CBR. In this section we discuss how to retrieve cases from a casebase to classify new instances. Having a casebase H as discovered by the CaseExtract algorithm, we can associate a new instance $t \in \mathbf{V}$ with a case $h \in H$ by checking whether t is covered by $\mathbb{E}$ through h. Then t is regarded as being in the same class as h. The CaseRetrieve algorithm is as follows.

- Sort the elements in H in decreasing order of $|X|$ for $h \in H$ and $h = \text{lub}(X)$, which results in $H = \{h_0, \cdots, h_n\}$ with h_0 having the largest coverage of $\mathbf{D}$ elements.
- t is classified by the first h_i in the sorted H such that the conditions in Theorem 4.1 are satisfied.
- If there is no such h_i, label t by $d(h_0)$.

The time complexity of this algorithm is dominated by calculating the R-complement of $h \in H$, as needed in Theorem 4.1. This is in the order of $O(T)$, where T is the number of attributes. In the worst case we need to do so for all h_i in H, $i = 0, \cdots, n$. Therefore the overall time complexity of the algorithm is $O(nT)$ in the worst case.

To illustrate the CaseRetrieve algorithm, consider Figure 1. The casebase is now $\{E, F, M\}$, as discovered by CaseExtract. Consider a new case G. The sum of G and E is B, which is equilabeled. Then E is retrieved

and G is labeled as positive. Clearly $G \leq B$, hence $G \preccurlyeq \mathbb{E}$.

7 Experiment

CaseExtract and CaseRetrieve are implemented in our CBR system, called LM. We compared LM with C4.5 using public datasets. The datasets are described in Table 2. Datasets in the upper half are from UC Irvine Machine Learning Repository; and those in the lower half are collections of documents which are used as benchmark for text mining study [Cohen and Hirsh, 1998]. The results are shown in Table 3.

Datasets	#Features #Terms	#Train	#Test	#Class
Annealing	38	798	5cv	6
Auto	25	205	5cv	6
Diabetes	8	768	5cv	2
Glass	9	214	5cv	6
Iris	4	150	5cv	3
Sonar	60	208	5cv	2
Vote	18	232	5cv	2
Memos	1014	334	10cv	11
CDroms	1133	798	10cv	6
Birdcom	674	914	10cv	22
Birdsci	1738	914	10cv	22

Table 2: Description of the datasets.

Dataset	Prediction accuracy	
	C4.5	LM
Annealing	91.8	93.6
Auto	72.2	76.1
Diabetes	72.9	71.7
Glass	81.3	82.7
Iris	94.0	96.0
Sonar	69.4	69.7
Vote	95.1	97.0
memos	57.5	59.8
cdroms	39.2	40.0
birdcom	79.6	90.4
birdsci	83.3	92.3
Average	76.0	79.0

Table 3: Prediction accuracy of C4.5 and LM.

8 Summary and conclusion

The paper proposed a promising model for automating the design of CBR systems. Revolving around the notion of hypertuples (abstract cases), the proposed model presents a successful attempt at combining powerful eager techniques from machine learning with the flexible "defer-processing" philosophy characteristic for lazy methods [Aha, 1997]. On the basis of concise formal argument and empirical evaluation, it has been demonstrated that the Lattice Machine approach constitutes an effective and efficient mechanism to discover abstract cases in a given dataset. Abstract cases have been shown to be an effective alternative to representing the knowledge held in CBR systems [Bergmann and Wilke, 1996]. They can provide answers to issues such as casebase complexity, maintenance costs, retrieval efficiency, and user acceptance. In addition to the discovery of abstract cases, an algorithm was presented, which employs the general boundary inductive bias and ensures that the retrieval of relevant abstract cases is within the limits of reasonable time constraints. The main contribution of this work lies in the Lattice Machine's ability to discover abstract cases within a given dataset *without* requiring difficult-to-obtain domain knowledge.

References

[Aha, 1997] D. Aha, editor. *Lazy Learning*. Kluver Academic Pub., 1997.

[Bergmann and Wilke, 1996] R. Bergmann and W. Wilke. On the role of abstraction in case-based reasoning. In *Proc. Advances in Case-Based Reasoning, 3rd EWCBR-96*, pages 28–41, 1996.

[Cohen and Hirsh, 1998] William W. Cohen and Haym Hirsh. Joins that generalize: Text classification using whirl. In *Proc. KDD-98, New York*, 1998.

[Grätzer, 1978] George Grätzer. *General Lattice Theory*. Birkhäuser, Basel, 1978.

[Haussler, 1988] D. Haussler. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[Leake and Wilson, 1998] D.B. Leake and D.C. Wilson. Categorizing case base maintenance: Dimensions and directions. In *Proc. Advances in Case-Based Reasoning,4th EWCBR-98*, pages 196–207, 1998.

[Leake, 1996] D.B. Leake, editor. *Case-Based Reasoning: Experiences, Lessons & Future Directions*. MIT Press, MA, 1996.

[Lenz *et al.*, 1998] M. Lenz, B. Bartsch-Spörl, H-D. Burkhard, and S. Wess, editors. *Case-Based Reasoning Technology: From Foundations to Applications*. Springer-Verlag, 1998.

[Mitchell, 1997] T. M. Mitchell. *Machine Learning*. The McGraw-Hill Companies, Inc, 1997.

[Orlowska, 1985] Ewa Orlowska. Logic of nondeterministic information. *Studia Logica*, 44:93–102, 1985.

[Patterson *et al.*, 1998] D. Patterson, W. Dubitzky, S.S. Anand, and J.G. Hughes. On the automation of case base development from large databases. In *Proc. AAAI Workshop: Case-Based Reasoning Integrations*, pages 126–130, 1998.

[Smyth and Keane, 1995] B. Smyth and M.T. Keane. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *Proc. 14th IJCAI-95*, pages 337–382, 1995.

[Wang *et al.*, 1998] Hui Wang, Ivo Düntsch, and David Bell. Data reduction based on hyper relations. In *Proceedings of KDD98, New York*, pages 349–353, 1998.