One-Sided Instance-Based Boundary Sets

Evgueni N. Smirnov, Ida G. Sprinkhuizen-Kuyper, and
H. Jaap van den Herik

IKAT, Department of Computer Science, Universiteit Maastricht,
P.O.BOX 616, 6200 MD Maastricht, The Netherlands
{smirnov, kuyper, herik}@cs.unimaas.nl

Abstract. Instance retraction is a difficult problem for concept learning by version spaces. This chapter introduces a family of version-space representations called one-sided instance-based boundary sets. They are correct and efficiently computable representations for admissible concept languages. Compared to other representations, they are the most efficient useful[1] version-space representations for instance retraction.

1 Introduction

Currently, there is a renewed interest in version spaces caused by their applicability in inductive databases [5,6]. This chapter considers version spaces when the inductive query constraints are instances of a concept to be learned; i.e., the task is essentially a concept-learning task. In this context we study two important problems of inductive databases: the problem of efficiently representing version spaces and the problem of efficiency of version spaces for instance retraction.

Mitchell defined version spaces as sets of concept descriptions that are consistent with training data [7,8]. Version-space learning is an incremental process:

- *If an instance i is added*, the version space is revised so that it consists of all the concept descriptions consistent with the processed training data *plus i*.
- *If an instance i is retracted*, the version space is revised so that it consists of all the concept descriptions consistent with the processed training data *minus i*.

For the learning processes version spaces are represented. The standard representation is the boundary-set representation [7,8]. It is correct for the class of admissible concept languages [7], but its size can grow exponentially in the size of training data [1]. To overcome this problem alternative version-space representations were introduced [2,3,4,9,10,11,12,13]. They extended the classes of concept languages for which version spaces are efficiently computable.

However a remaining problem for most version-space representations is that they are inefficient for instance retraction, since they lack a structure that determines the influence of an individual training instance. Hence, if a training instance is retracted, the representations are recomputed [11]. To avoid this problem two version-space representations were proposed. The first one is the

[1] In this chapter, the notion useful has a technical meaning defined in subsection 2.2.

R. Meo et al. (Eds.): Database Support for Data Mining Applications, LNAI 2682, pp. 270–288, 2004.

training-instance representation [3]. By its definition it is efficient for instance retraction. However, the representation has only a theoretical value, since the classification of each instance requires search in the concept language using all the training data. The second representation is instance-based boundary sets (IBBS) [11,12]. It is correct and efficiently computable for the class of admissible concept languages. The instance-retraction algorithm of the IBBS is efficient and it does not recompute the representation. At the moment the IBBS is the most efficient useful version-space representation for instance retraction.

In this chapter we address the question whether it is possible to design new version-space representations that are even more efficient than the IBBS in terms of computability and instance retraction. To answer the question we introduce a family of version-space representations called one-sided instance-based boundary sets. The family consists of two dual representations: instance-based maximal boundary sets (IBMBS) and instance-based minimal boundary sets (IBmBS). Without loss of generality we consider in detail only the IBMBS representation.

The course of the chapter is as follows. In section 2 we formalise the necessary basic notions. They are used in section 3 to define the IBMBS representation. There, we prove that the representation is correct for the class of admissible concept languages and derive the conditions for finiteness. Section 4 presents four IBMBS algorithms for instance addition, instance retraction, version-space collapse and instance classification. It is shown that the IBMBS can be used for instance classification in the presence of noisy training data. In sections 5 and 6 we provide an analysis and an evaluation of usefulness of the IBMBS. The dual representation of the IBMBS, instance-based minimal boundary sets (IBmBS), is touched upon in section 7. We compare the new representations with relevant work in section 8. Finally, in section 9 conclusions are given.

2 Formalisation

This section formalises the necessary basic notions. In subsection 2.1 we formulate the concept-learning task. Then, in subsection 2.2, we introduce version spaces as a solution of the task and we consider the notion of version-space representations together with their characteristics of usefulness. In this context we present the class of admissible concept languages in subsection 2.3.

2.1 The Concept-Learning Task

Concept learning assumes the presence of a universe of all the instances [11].

Notation 1. The universe of all the instances is denoted by I.

Definition 2 (Concept). *A concept* $\mathfrak{c}$ *is subset of* I*:* $\mathfrak{c} \subseteq I$.

Given a concept, there exist two types of instance sets.

Definition 3 (Set of Positive/Negative Instances). *A set* $I^+ \subseteq I$ *is a set of positive instances of a concept* $\mathfrak{c} \subseteq I$ *if and only if* $I^+ \subseteq \mathfrak{c}$*. A set* $I^- \subseteq I$ *is a set of negative instances of a concept* $\mathfrak{c} \subseteq I$ *if and only if* $I^- \cap \mathfrak{c} = \emptyset$.

The set of all the concepts defined on the universe I is the power set $P(I)$. To represent concepts from $P(I)$ we introduce a language.

Definition 4 (Concept Language). *The concept language Lc is a set of descriptions c.*

To associate a description $c \in Lc$ with a concept $\mathfrak{c} \in P(I)$ that c represents, we define a function $\mathcal{R}_{\mathfrak{c}}$.

Definition 5. *The function $\mathcal{R}_{\mathfrak{c}} : Lc \rightarrow P(I)$ is an injective function that maps a concept description $c \in Lc$ to a concept $\mathfrak{c} \in P(I)$.*

Since $\mathcal{R}_{\mathfrak{c}}$ is a function, no two distinct concepts in $P(I)$ can be represented by the same description in Lc. Since $\mathcal{R}_{\mathfrak{c}}$ is injective, no two distinct descriptions in Lc can represent the same concept in $P(I)$.

Instances are related to concepts by the membership relation. The relation is projected into a cover relation between instances and concept descriptions [7].

Definition 6 (Cover Relation M).
$M : Lc \times I \rightarrow Boolean$
defined by: $M(c, i) \leftrightarrow i \in \mathcal{R}_{\mathfrak{c}}(c)$.

The cover relation M holds for a description $c \in Lc$ and an instance $i \in I$ if and only if i is a member of the concept $\mathcal{R}_{\mathfrak{c}}(c)$. If the relation M holds for $c \in Lc$ and $i \in I$, we say that c covers i; otherwise, we say that c does not cover i.

After the introduction of the elements of the concept-learning task we formulate the task itself according to [7,8,11]. *Given a universe I of all the instances, a concept language Lc, a cover relation M, and the training sets I^+ and I^- of a target concept, the task is to find descriptions of the target concept in Lc.*

2.2 Version Spaces

A version space is a solution of the concept-learning task. It is a set of all the concept descriptions that are consistent with the training sets I^+ and I^- [7,8]. A description $c \in Lc$ is consistent with the sets I^+ and I^- if and only if c covers each instance $p \in I^+$ and does not cover any instance $n \in I^-$. Below we give a formal definition of version spaces.

Definition 7 (Version Space). *Given the training sets I^+ and I^- of a target concept, the version space $VS(I^+, I^-)$ is defined as follows:*

$$VS(I^+, I^-) = \{c \in Lc \mid (\forall p \in I^+)M(c, p) \wedge (\forall n \in I^-)\neg M(c, n)\}.$$

To learn version spaces we need a representation. A version-space representation is a structure that contains "information needed to reconstruct every description in the version space" [7]. It has four possible characteristics: (1) compactness; (2) finiteness; (3) efficient computability [2,11]; and (4) efficiency of the algorithms for instance addition, instance retraction, version-space collapse and instance classification. To encapsulate these characteristics we introduce the notion of a useful version-space representation.

Definition 8 (Useful Version-Space Representation). *A version-space representation is useful if and only if it is compact, finite, efficiently computable and has efficient algorithms for instance addition, instance retraction, version-space collapse, and instance classification.*

2.3 Admissible Concept Languages

The key to find a compact version-space representation is to observe that concept languages can be ordered. This can be done by a partially-ordering relation "more general". The relation is taken from [7,8] and is defined below.

Definition 9 (Relation "More General" ($\geq$)).

$$(\forall c_1, c_2 \in Lc)((c_1 \geq c_2) \leftrightarrow (\forall i \in I)(M(c_1, i) \leftarrow M(c_2, i))).$$

A description $c_1 \in Lc$ is more general than a description $c_2 \in Lc$ ($c_1 \geq c_2$) if and only if for each instance $i \in I$ if c_2 covers i, then c_1 covers i as well. If a description $c_1 \in Lc$ is more general than a description $c_2 \in Lc$ we say that c_1 is a generalisation of c_2 and c_2 is a specialisation of c_1.

If the relation "$\geq$" is defined on a concept language Lc, then Lc is partially-ordered. For defining version-space representations one class of partially-ordered languages was extensively used, viz. the class of admissible concept languages [7,11]. It is introduced after the definition of minimal and maximal sets of a partially-ordered set (cf. [7]).

Definition 10 (Minimal/Maximal Set). *If C is a partially-ordered set, then:*

$$MIN(C) = \{c \in C | (\forall c' \in C)((c \geq c') \rightarrow (c' = c))\}$$
$$MAX(C) = \{c \in C | (\forall c' \in C)((c' \geq c) \rightarrow (c' = c))\}.$$

A partially-ordered concept language Lc is admissible if each subset of Lc is bounded. A partially-ordered set is bounded if all the elements of the set are between its minimal and maximal elements. Below we give a formal definition.

Definition 11 (Admissible Concept Language). *A partially-ordered concept language Lc is admissible if and only if for every nonempty subset $C \subseteq Lc$:*

$$C \subseteq \{c \in Lc | (\exists s \in MIN(C))(c \geq s) \wedge (\exists g \in MAX(C))(g \geq c)\}.$$

Given a version space $VS(I^+, I^-)$ in an admissible concept language, the maximal set of $VS(I^+, I^-)$ is known as the maximal boundary set of $VS(I^+, I^-)$ and the minimal set of $VS(I^+, I^-)$ as the minimal boundary set of $VS(I^+, I^-)$.

Notation 12. The maximal boundary set and the minimal boundary set of version space $VS(I^+, I^-)$ are denoted by $G(I^+, I^-)$ and $S(I^+, I^-)$, respectively.

3 Instance-Based Maximal Boundary Sets

Below we introduce instance-based maximal boundary sets (IBMBS) as a new version-space representation. The correctness of the representation is proven for the class of admissible concept languages. The IBMBS are shown to be compact and their conditions for finiteness are derived.

3.1 Definition and Correctness

The IBMBS representation consists of the set of positive training instances and a family of maximal boundary sets indexed by negative training instances. Below the representation is formally defined.

Definition 13 (Instance-Based Maximal Boundary Sets). *Consider an admissible concept language Lc and training sets $I^+ \subseteq I$ and $I^- \subseteq I$ so that $I^- \neq \emptyset$. Then the instance-based-maximal-boundary-set representation of a version space $VS(I^+, I^-)$ is the ordered pair $\langle I^+, \{G(I^+, \{n\})\}_{n \in I^-} \rangle$.*

The IBMBS are "instance-based" since each of their elements corresponds to particular training instances. The IBMBS are "maximal boundary sets" since each of their elements in $\{G(I^+, \{n\})\}_{n \in I^-}$ is a maximal boundary set. The IBMBS are a one-sided version space representation since its first part is the set I^+; i.e., this part does not contain any boundary-set element.

To prove that the IBMBS are a correct version-space representation we give theorems 14 and 15 from [11]. Theorem 14 states that if a description $c \in Lc$ is more specific than at least one element of each maximal boundary set $G(I^+, \{n\})$ for all $n \in I^-$, then c is consistent with the set I^- of negative instances.

Theorem 14. *If the concept language Lc is admissible, then:*

$$(\forall c \in Lc)((\forall n \in I^-)(\exists g \in G(I^+, \{n\}))(g \geq c) \rightarrow (\forall n \in I^-)\neg M(c, n)).$$

Theorem 15 states that a version space $VS(I_1^+, I_1^-)$ is a subset of a version space $VS(I_2^+, I_2^-)$ if and only if every description in $VS(I_1^+, I_1^-)$ is consistent with the sets I_2^+ and I_2^-.

Theorem 15. $VS(I_1^+, I_1^-) \subseteq VS(I_2^+, I_2^-) \leftrightarrow$
$(\forall c \in VS(I_1^+, I_1^-))((\forall p \in I_2^+)M(c, p) \wedge (\forall n \in I_2^-)\neg M(c, n)).$

Theorem 16 (Correctness of IBMBS). *Consider a version space $VS(I^+, I^-)$ represented by IBMBS: $\langle I^+, \{G(I^+, \{n\})\}_{n \in I^-} \rangle$. If the concept language Lc is admissible, then:*

$$(\forall c \in Lc)(c \in VS(I^+, I^-) \leftrightarrow ((\forall p \in I^+)M(c, p) \wedge (\forall n \in I^-)(\exists g \in G(I^+, \{n\}))(g \geq c))).$$

Proof. ($\rightarrow$) Consider an arbitrarily chosen description $c \in VS(I^+, I^-)$. By theorem 15 $(\forall n \in I^-)(VS(I^+, I^-) \subseteq VS(I^+, \{n\}))$. Thus, $(\forall n \in I^-)(c \in VS(I^+, \{n\}))$. Since Lc is admissible, according to definition 11 for each $VS(I^+, \{n\})$ we have:

$$(\forall n \in I^-)(\exists g \in G(I^+, \{n\}))(g \geq c). \tag{1}$$

Since $c \in VS(I^+, I^-)$, according to definition 7:

$$(\forall p \in I^+)M(c, p). \tag{2}$$

From (1) and (2) the first part of theorem is proven.
($\leftarrow$) Let $c \in Lc$ be arbitrarily chosen so that:

$$(\forall p \in I^+)M(c,p). \tag{3}$$
$$(\forall n \in I^-)(\exists g \in G(I^-,\{n\}))(g \geq c). \tag{4}$$

By theorem 14 formula (4) implies:

$$(\forall n \in I^-)\neg M(c,n). \tag{5}$$

Thus, $c \in Lc$, (3), and (5) imply $c \in VS(I^+, I^-)$ according to definition 7. □

Given the IBMBS of a version space $VS(I^+, I^-)$ and an admissible concept language, theorem 16 states that the concept descriptions in $VS(I^+, I^-)$ are exactly those that (1) cover all the positive instances in I^+, and (2) are more specific than at least one element of each maximal boundary set $G(I^+, \{n\})$. This means that the size of IBMBS is not tied to the number of descriptions in the version space. Thus, the IBMBS are a compact version-space representation.

Example 1. Let the instance universe I and the concept language Lc be 1-CNF languages with 8 attributes. The domain of the k-th attribute in I is $\{0,1\}$ and in Lc is $\{0,1,?\}$, where the symbol "?" indicates that any value is acceptable. The procedure of the cover relation M returns true for a concept description $c \in Lc$ and an instance $i \in I$ if and only if for each attribute the values of c and i are equal or the value of c equals "?". In this context we consider a concept-learning task with the set I^+ consisting of one positive instance: $i_1^+ = \langle 1,1,1,1,1,1,1,1\rangle$ and the set I^- consisting of three negative instances: $i_2^- = \langle 0,0,1,1,1,1,1,1\rangle$, $i_3^- = \langle 1,1,0,0,1,1,1,1\rangle$ and $i_4^- = \langle 1,1,1,1,0,0,1,1\rangle$. For this task, the IBMBS: $\langle I^+, \{G(I^+,\{n\})\}_{n\in I^-}\rangle$ of the version space $VS(I^+, I^-)$ consist of four sets:

$$\begin{aligned}
I^+ &= \{\langle 1,1,1,1,1,1,1,1\rangle\},\\
G(I^+,\{i_2^-\}) &= \{\langle 1,?,?,?,?,?,?,?\rangle, \langle ?,1,?,?,?,?,?,?\rangle\},\\
G(I^+,\{i_3^-\}) &= \{\langle ?,?,1,?,?,?,?,?\rangle, \langle ?,?,?,1,?,?,?,?\rangle\},\\
G(I^+,\{i_4^-\}) &= \{\langle ?,?,?,?,1,?,?,?\rangle, \langle ?,?,?,?,?,1,?,?\rangle\}.
\end{aligned}$$

□

3.2 Finiteness

Since the IBMBS are compact, it is important to determine when they are finite. We introduce constraints on the training sets and the concept language. We show that they are sufficient and necessary conditions for the finiteness of the IBMBS.

We start with the constraints on the training sets.

Constraint 17. The training sets I^+ and I^- are finite.

Constraint 17 implies that the number of the maximal boundary sets $G(I^+, \{n\})$ is finite. Hence, IBMBS are finite in this case if each set $G(I^+, \{n\})$ is finite. To guarantee this property we introduce a constraint on the concept language.

Constraint 18. The maximal boundary set $G(\emptyset, \{n\})$ is finite for all $n \in I$.

To explain how constraint 18 affects each maximal boundary set $G(I^+, \{n\})$ theorem 19 is taken from [7]. The theorem states that the set $G(I^+ \cup \{i\}, I^-)$ is equal to the set of those elements of the set $G(I^+, I^-)$ that cover the instance i.

Theorem 19. *Consider training sets $I^+, I^- \subseteq I$. If the concept language Lc is admissible, then for all $p \in I$:*

$$G(I^+ \cup \{p\}, I^-) = \{g \in G(I^+, I^-) | M(g, p)\}.$$

An important consequence of theorem 19 is given in corollary 20 below.

Corollary 20. *Consider sets $I_1^+, I_2^+ \subseteq I$ so that $I_2^+ \subseteq I_1^+$. Then for all $n \in I$:*

$$G(I_1^+, \{n\}) \subseteq G(I_2^+, \{n\}).$$

Using corollary 20 we formulate the following theorem.

Theorem 21. *The maximal boundary set $G(I^+, \{n\})$ is finite for all $n \in I$ and $I^+ \subseteq I$ if and only if constraint 18 holds.*

Combining constraints 17 and 18, and using theorem 21 we finish the section by formulating the theorem of the IBMBS being finite.

Theorem 22. *The IBMBS are finite if and only if constraints 17 and 18 hold.*

4 Algorithms of the IBMBS

This section introduces four algorithms of the IBMBS. The instance-addition algorithm is given in subsection 4.1; the instance-retraction algorithm is given in subsection 4.2; the algorithm for version-space collapse is given in subsection 4.3; and the instance-classification algorithm and its extension for noisy training data are given in subsection 4.4.

4.1 Instance-Addition Algorithm

The instance-addition algorithm of the IBMBS revises the representation given a new training instance. It is correct for the class of admissible concept languages. The algorithm consists of two parts for handling positive and negative training instances. They are based on theorem 23 and theorem 16, respectively.

Theorem 23. *Consider a version space $VS(I^+, I^-)$ represented by IBMBS: $\langle I^+, \{G(I^+, \{n\})\}_{n \in I^-} \rangle$, and a version space $VS(I^+ \cup \{i\}, I^-)$ represented by IBMBS: $\langle I^+ \cup \{i\}, \{G(I^+ \cup \{i\}, \{n\})\}_{n \in I^-} \rangle$. If the concept language Lc is admissible, then:*

$$G(I^+ \cup \{i\}, \{n\}) = \{g \in G(I^+, \{n\}) \,|\, M(g, i)\} \text{ for all } n \in I^-.$$

Proof. The theorem follows from theorem 19. □

Instance-Addition **Algorithm**
Input: i: a new training instance.
$\langle\{I^+, \{G(I^+, \{n\})\}_{n\in I^-}\rangle$: IBMBS of $VS(I^+, I^-)$.
Output:
$\langle I^+ \cup \{i\}, \{G(I^+ \cup \{i\}, \{n\})\}_{n\in I^-}\rangle$: IBMBS of $VS(I^+ \cup \{i\}, I^-)$ if i is positive.
$\langle I^+, \{G(I^+, \{n\})\}_{n\in I^-\cup\{i\}}\rangle$: IBMBS of $VS(I^+, I^- \cup \{i\})$ if i is negative.
Precondition: Lc is admissible.

if instance i is positive **then**
for $n \in I^-$ **do**
$G(I^+ \cup \{i\}, \{n\}) = \{g \in G(I^+, \{n\}) \mid M(g, i)\}$
return $\langle I^+ \cup \{i\}, \{G(I^+ \cup \{i\}, \{n\})\}_{n\in I^-}\rangle$
if instance i is negative **then**
Generate the set $G(\emptyset, \{i\})$
$G(I^+, \{i\}) = \{g \in G(\emptyset, \{i\}) | (\forall p \in I^+) M(g, p)\}$
return $\langle\{I^+, \{G(I^+, \{n\})\}_{n\in I^-\cup\{i\}}\rangle$.

Fig. 1. The Instance-Addition Algorithm

The instance-addition algorithm can be described as follows (see in figure 1). If a new positive training instance i is given, the algorithm forms the maximal boundary sets $G(I^+\cup\{i\}, \{n\})$ for all $n \in I^-$. Each set $G(I^+\cup\{i\}, \{n\})$ is formed from those elements of the corresponding set $G(I^+, \{n\})$ that cover the instance i. The resulting IBMBS of the version space $VS(I^+ \cup \{i\}, I^-)$ are formed from the set $I^+ \cup \{i\}$ and the maximal boundary sets $G(I^+ \cup \{i\}, \{n\})$ for all $n \in I^-$.

If the instance i is negative, the algorithm first forms the maximal boundary set $G(I^+, \{i\})$ in two steps. In the first step it generates the maximal boundary set $G(\emptyset, \{i\})$. In the second step the algorithm forms $G(I^+, \{i\})$ from the elements of $G(\emptyset, \{i\})$ that cover all the instances in I^+ (see theorem 19). Then, the resulting IBMBS of the version space $VS(I^+, I^- \cup \{i\})$ are formed from the set I^+ and the maximal boundary sets $G(I^+, \{n\})$ for all $n \in I^- \cup \{i\}$.

Example 2. Let us illustrate the instance-addition algorithm given the IBMBS from example 1. Assume that we have a new negative training instance $i_5^- = \langle 1, 1, 1, 1, 1, 1, 0, 0\rangle$. The algorithm first generates the maximal boundary set $G(I^+, \{i_5^-\}) = \{\langle ?, ?, ?, ?, ?, ?, 1, ?\rangle, \langle ?, ?, ?, ?, ?, ?, ?, 1\rangle\}$. Then, it adds $G(I^+, \{i_5^-\})$ to the IBMBS. The resulting IBMBS consist of five sets:

$$I^+ = \{\langle 1, 1, 1, 1, 1, 1, 1, 1\rangle\},$$
$$G(I^+, \{i_2^-\}) = \{\langle 1, ?, ?, ?, ?, ?, ?, ?\rangle, \langle ?, 1, ?, ?, ?, ?, ?, ?\rangle\},$$
$$G(I^+, \{i_3^-\}) = \{\langle ?, ?, 1, ?, ?, ?, ?, ?\rangle, \langle ?, ?, ?, 1, ?, ?, ?, ?\rangle\},$$
$$G(I^+, \{i_4^-\}) = \{\langle ?, ?, ?, ?, 1, ?, ?, ?\rangle, \langle ?, ?, ?, ?, ?, 1, ?, ?\rangle\},$$
$$G(I^+, \{i_5^-\}) = \{\langle ?, ?, ?, ?, ?, ?, 1, ?\rangle, \langle ?, ?, ?, ?, ?, ?, ?, 1\rangle\}.$$

Assume now that we have a new positive instance $i_6^+ = \langle 1, 0, 1, 0, 1, 0, 1, 0\rangle$. The algorithm forms for each $n \in I^-$ the maximal boundary set $G(I^+\cup\{i_6^+\}, \{n\})$

from the elements of the set $G(I^+, \{n\})$ that cover the instance i_6^+. It adds the instance i_6^+ to the training set I^+. The resulting IBMBS consist of five sets:

$$I^+ = \{\langle 1,1,1,1,1,1,1,1\rangle, \langle 1,0,1,0,1,0,1,0\rangle\},$$
$$G(I^+, \{i_2^-\}) = \{\langle 1,?,?,?,?,?,?,?\rangle\}, G(I^+, \{i_3^-\}) = \{\langle ?,?,1,?,?,?,?,?\rangle\},$$
$$G(I^+, \{i_4^-\}) = \{\langle ?,?,?,?,1,?,?,?\rangle\}, G(I^+, \{i_5^-\}) = \{\langle ?,?,?,?,?,?,1,?\rangle\}. \quad \square$$

4.2 Instance-Retraction Algorithm

The instance-retraction algorithm of the IBMBS revises the representation when an instance is removed from one of the training sets. It is correct for the class of admissible concept languages when the property G holds [11].

Definition 24 (Property G). *An admissible concept language is said to have property G if for all $n_1, n_2 \in I$:*

$$\{g \in G(\emptyset, \{n_1\}) | \neg M(g, n_2)\} = \{g \in G(\emptyset, \{n_2\}) | \neg M(g, n_1)\}.$$

An admissible concept language has the property G if for all $n_1, n_2 \in I$ the subset of the elements of the set $G(\emptyset, \{n_1\})$, that do not cover the instance n_2, equals the subset of the elements of the set $G(\emptyset, \{n_2\})$, that do not cover the instance n_1. A simple consequence of the property G is given in a corollary below.

Corollary 25. *If the property G holds, then for all $n_1, n_2 \in I$, and all $I^+ \subseteq I$:*

$$\{g \in G(I^+, \{n_1\}) | \neg M(g, n_2)\} = \{g \in G(I^+, \{n_2\}) | \neg M(g, n_1)\}.$$

The instance-retraction algorithm consists of two parts for handling positive and negative instances. They are based on theorems 26 and 16, respectively.

Theorem 26. *Consider a version space $VS(I^+, I^-)$ represented by IBMBS: $\langle I^+, \{G(I^+, \{n\})\}_{n \in I^-}\rangle$, and a second version space $VS(I^+ \setminus \{i\}, I^-)$ represented by IBMBS: $\langle I^+ \setminus \{i\}, \{G(I^+ \setminus \{i\}, \{n\})\}_{n \in I^-}\rangle$, where $i \in I^+$. If the concept language Lc is admissible and the property G holds, then:*

$$G(I^+ \setminus \{i\}, \{n\}) = G(I^+, \{n\}) \cup \{g \in G(I^+ \setminus \{i\}, \{i\}) | \neg M(g, n)\} \text{ for all } n \in I^-.$$

Proof. For each $n \in I^-$:

$$G(I^+ \setminus \{i\}, \{n\}) = \{g \in G(I^+ \setminus \{i\}, \{n\}) | M(g, i)\} \cup \{g \in G(I^+ \setminus \{i\}, \{n\}) | \neg M(g, i)\}.$$

According to theorem 19:

$$\{g \in G(I^+ \setminus \{i\}, \{n\}) | M(g, i)\} = G(I^+, \{n\})$$

and according to corollary 25:

$$\{g \in G(I^+ \setminus \{i\}, \{n\}) | \neg M(g, i)\} = \{g \in G(I^+ \setminus \{i\}, \{i\}) | \neg M(g, n)\}.$$

Thus,

$$G(I^+ \setminus \{i\}, \{n\}) = G(I^+, \{n\}) \cup \{g \in G(I^+ \setminus \{i\}, \{i\}) | \neg M(g, n)\}. \quad \square$$

Instance-Retraction **Algorithm**
Input: i: a training instance in $I^+ \cup I^-$.
$\langle \{I^+, \{G(I^+, \{n\})\}_{n \in I^-} \rangle$: IBMBS of $VS(I^+, I^-)$.
Output: $\langle \{I^+ \setminus \{i\}, \{G(I^+ \setminus \{i\}, \{n\})\}_{n \in I^-} \rangle$: IBMBS of $VS(I^+ \setminus \{i\}, I^-)$ if $i \in I^+$.
$\langle \{I^+, \{G(I^+, \{n\})\}_{n \in I^- \setminus \{i\}} \rangle$: IBMBS of $VS(I^+, I^- \setminus \{i\})$ if $i \in I^-$.
Precondition: Lc is admissible, property G holds, and $|I^-| > 1$ if $i \in I^-$.

if $i \in I^+$ **then**
Generate the set $G(\emptyset, \{i\})$
$G(I^+ \setminus \{i\}, \{i\}) = \{g \in G(\emptyset, \{i\}) | (\forall p \in I^+ \setminus \{i\}) M(g, p)\}$
for $n \in I^-$ **do**
$G(I^+ \setminus \{i\}, \{n\}) = G(I^+, \{n\}) \cup \{g \in G(I^+ \setminus \{i\}, \{i\}) \,|\, \neg M(g, n)\}$
return $\langle I^+ \setminus \{i\}, \{G(I^+ \setminus \{i\}, \{n\})\}_{n \in I^-} \rangle$
if $i \in I^-$ **then**
return $\langle I^+, \{G(I^+, \{n\})\}_{n \in I^- \setminus \{i\}} \rangle$.

Fig. 2. The Instance-Retraction Algorithm

The instance-retraction algorithm can be described as follows (see figure 2). If an instance i is removed from the set I^+, the algorithm executes two steps following theorem 26. In the first step it forms the maximal boundary set $G(I^+ \setminus \{i\}, \{i\})$. This is done by first generating the maximal boundary set $G(\emptyset, \{i\})$ and then by removing those elements of $G(\emptyset, \{i\})$ that do not cover at least one instance in $I^+ \setminus \{i\}$ (see theorem 19). In the second step the algorithm forms the maximal boundary set $G(I^+ \setminus \{i\}, \{n\})$ for each $n \in I^-$. The set $G(I^+ \setminus \{i\}, \{n\})$ is formed as a union of the corresponding sets $G(I^+, \{n\})$ and $\{g \in G(I^+ \setminus \{i\}, \{i\}) \,|\, \neg M(g, n)\}$. The resulting IBMBS of the version space $VS(I^+ \setminus \{i\}, I^-)$ are formed from the set $I^+ \setminus \{i\}$ and the maximal boundary sets $G(I^+ \setminus \{i\}, \{n\})$ for all $n \in I^-$.

If the instance i is removed from the set I^-, the algorithm forms the resulting IBMBS of the version space $VS(I^+, I^- \setminus \{i\})$ from the set I^+ and the maximal boundary sets $G(I^+, \{n\})$ for all $n \in I^- \setminus \{i\}$.

Example 3. Let us illustrate the instance-retraction algorithm given the last IBMBS from example 2. Note that the property G holds for the concept language used. Assume that we have to retract the positive training instance $i_6^+ = \langle 1, 0, 1, 0, 1, 0, 1, 0 \rangle$. The algorithm forms the boundary set $G(I^+ \setminus \{i_6^+\}, \{i_6^+\}) = \{\langle ?, 1, ?, ?, ?, ?, ?, ? \rangle,\ \langle ?, ?, ?, 1, ?, ?, ?, ? \rangle,\ \langle ?, ?, ?, ?, ?, 1, ?, ? \rangle,\ \langle ?, ?, ?, ?, ?, ?, ?, 1 \rangle\}$. Then, it forms for each $n \in I^-$ the maximal boundary set $G(I^+ \setminus \{i_6^+\}, \{n\})$ as a union of the sets $G(I^+, \{n\})$ and $\{g \in G(I^+ \setminus \{i_6^+\}, \{i_6^+\}) \,|\, \neg M(g, n)\}$. The instance i_6^+ is excluded from the training set I^+ and the resulting IBMBS coincide with the first IBMBS from example 2.

Assume now that we have to retract the negative training instance $i_5^- = \langle 1, 1, 1, 1, 1, 1, 0, 0 \rangle$. The algorithm excludes: (1) the instance from the training set I^-, and (2) the maximal boundary set $G(I^+, \{i_5\})$ from the IBMBS. The resulting IBMBS coincide with those from example 1. □

4.3 Algorithm for Version-Space Collapse

The algorithm for version-space collapse checks whether a version space represented by IBMBS is empty. It is proposed for the class of admissible concept languages when the intersection-preserving property (IP) holds [11].

Definition 27 (Intersection-Preserving Property (IP)). *An admissible concept language is said to have the intersection-preserving property if for each nonempty set $C \subseteq Lc$ there exists a description $c \in Lc$ so that:*

$$(\forall i \in I)((\forall c' \in C)M(c', i) \leftrightarrow M(c, i)).$$

An admissible concept language Lc exhibits the property IP when for each nonempty subset $C \subseteq Lc$ there exists a description $c \in Lc$ so that an instance $i \in I$ is covered by all the descriptions $c' \in C$ if and only if i is covered by c. The property is introduced because it guarantees that if the training set I^- is not empty, the version space $VS(I^+, I^-)$ is not empty if and only if for each $n \in I^-$ the version space $VS(I^+, \{n\})$ is not empty (see theorem 28 taken from [11]).

Theorem 28. *Consider an admissible concept language Lc such that the property IP holds. If the set I^- is nonempty, then:*

$$(VS(I^+, I^-) \neq \emptyset) \leftrightarrow (\forall n \in I^-)(VS(I^+, \{n\}) \neq \emptyset).$$

To check a version space $VS(I^+, I^-)$ for collapse, by theorem 28 we can check for collapse of the version spaces $VS(I^+, \{n\})$ for $n \in I^-$. Since $VS(I^+, \{n\})$ are given by maximal boundary sets $G(I^+, \{n\})$ in the IBMBS of $VS(I^+, I^-)$, we give a relation between the sets $G(I^+, \{n\})$ and version spaces $VS(I^+, \{n\})$ [11].

Theorem 29. $(VS(I^+, I^-) \neq \emptyset) \leftrightarrow (G(I^+, I^-) \neq \emptyset)$.

Theorems 28 and 29 imply corollary 30 below. It states that if the property IP holds and the training set I^- is nonempty, the version space $VS(I^+, I^-)$ is nonempty if and only if for each $n \in I^-$ the set $G(I^+, \{n\})$ is nonempty.

Corollary 30. *Consider an admissible concept language Lc such that the property IP holds. If the set I^- is nonempty, then:*

$$(VS(I^+, I^-) \neq \emptyset) \leftrightarrow (\forall n \in I^-)(G(I^+, \{n\}) \neq \emptyset).$$

The version-space collapse algorithm is given in figure 3. If a version space $VS(I^+, I^-)$, given by IBMBS, is checked for collapse, the algorithm visits the maximal boundary sets $G(I^+, \{n\})$ for $n \in I^-$. If none of the sets $G(I^+, \{n\})$ is empty, by corollary 30 $VS(I^+, I^-)$ is not empty and the algorithm returns false. Otherwise, by corollary 30 $VS(I^+, I^-)$ is empty and the algorithm returns true.

Example 4. Let us illustrate the algorithm for version-space collapse given the IBMBS from example 1. Note that property IP holds for the concept language used. The algorithm checks the maximal boundary sets $G(I^+, \{i_2^-\})$, $G(I^+, \{i_3^-\})$, and $G(I^+, \{i_4^-\})$. Since none of them is empty, the algorithm returns false; i.e., the version space is nonempty. □

VS-Collapse **Algorithm**
Input: $\langle \{I^+, \{G(I^+, \{n\})\}_{n \in I^-} \rangle$: IBMBS of $VS(I^+, I^-)$.
Output: true if $VS(I^+, I^-) = \emptyset$.
false if $VS(I^+, I^-) \neq \emptyset$.
Precondition: Lc is admissible and the property IP holds.

for $n \in I^-$ **do**
if $G(I^+, \{n\}) = \emptyset$ **then**
return true
return false.

Fig. 3. The Algorithm for Version-Space Collapse

4.4 Instance-Classification Algorithm

Instance classification with version spaces is realised by the unanimous-voting rule: an instance is classified if and only if all the descriptions in a version space agree on a classification of the instance [7,8]. The rule can be implemented using theorems 31 and 32 taken from [11]. Theorem 31 states that all the descriptions of a version space $VS(I^+, I^-)$ do cover an instance $i \in I$ if and only if the version space $VS(I^+, I^- \cup \{i\})$ is empty. Theorem 32 states that all the descriptions of a version space $VS(I^+, I^-)$ do not cover an instance $i \in I$ if and only if the version space $VS(I^+ \cup \{i\}, I^-)$ is empty.

Theorem 31. $(\forall i \in I)((\forall c \in VS(I^+, I^-))M(c, i) \leftrightarrow (VS(I^+, I^- \cup \{i\}) = \emptyset))$.

Theorem 32. $(\forall i \in I)((\forall c \in VS(I^+, I^-))\neg M(c, i) \leftrightarrow (VS(I^+ \cup \{i\}, I^-) = \emptyset))$.

The instance-classification algorithm of the IBMBS realises the unanimous-voting rule for the class of admissible concept languages if the property IP holds. The positive instance classification is based on theorem 31, and the negative instance classification is based on theorem 33. Theorem 33 is used instead of theorem 32 for efficiency reasons. It states that if the concept language is admissible and the property IP holds, then all the descriptions of a version space $VS(I^+, I^-)$ do not cover an instance $i \in I$ if and only if there exists a version space $VS(I^+, \{n\})$ of which all the descriptions do not cover the instance i.

Theorem 33. *Consider an admissible concept language Lc such that the property IP holds. If the set I^- is nonempty, then:*

$(\forall i \in I)((\forall c \in VS(I^+, I^-))\neg M(c, i) \leftrightarrow (\exists n \in I^-)(\forall c \in VS(I^+, \{n\}))\neg M(c, i))$.

Proof. Consider an arbitrary $i \in I$. Then:

$(\forall c \in VS(I^+, I^-))\neg M(c, i)$ iff (theorem 32)
$VS(I^+ \cup \{i\}, I^-) = \emptyset$ iff (theorem 28)
$(\exists n \in I^-)\, VS(I^+ \cup \{i\}, \{n\}) = \emptyset$ iff (theorem 32)
$(\exists n \in I^-)(\forall c \in VS(I^+, \{n\}))\neg M(c, i)$ □

By theorem 33 a negative instance classification can be obtained by the version spaces $VS(I^+, \{n\})$. Since $VS(I^+, \{n\})$ are given with the maximal boundary sets $G(I^+, \{n\})$ in the IBMBS of $VS(I^+, I^-)$, we show how to use these sets for the classification using theorem 34 from [11].

Theorem 34. $(\forall i \in I)((\forall c \in VS(I^+, I^-))\neg M(c, i) \leftrightarrow (\forall g \in G(I^+, I^-))\neg M(g, i))$.

Theorems 33 and 34 imply corollary 35 below. Corollary 35 states that if an admissible concept language has the property IP, then none of the descriptions of a version space $VS(I^+, I^-)$ covers an instance $i \in I$ if and only if there exists a set $G(I^+, \{n\})$ of which all the descriptions do not cover the instance i.

Corollary 35. *Consider an admissible concept language Lc such that the property IP holds. If the set I^- is nonempty, then:*

$$(\forall i \in I)((\forall c \in VS(I^+, I^-))\neg M(c, i) \leftrightarrow (\exists n \in I^-)(\forall g \in G(I^+, \{n\}))\neg M(g, i)).$$

The instance-classification algorithm of the IBMBS is shown in figure 4. Given a nonempty version space $VS(I^+, I^-)$, it classifies an instance $i \in I$ in two steps. In the first step the algorithm forms the IBMBS of the version space $VS(I^+, I^- \cup \{i\})$ using the instance-addition algorithm applied on the IBMBS of $VS(I^+, I^-)$ with the instance i labeled as negative. If $VS(I^+, I^- \cup \{i\})$ is empty, by theorem 31 all the descriptions in $VS(I^+, I^-)$ cover the instance. Hence, the instance i is positive and the algorithm returns "+". If $VS(I^+, I^- \cup \{i\})$ is not empty, during the second step the algorithm visits the sets $G(I^+, \{n\})$ for $n \in I^-$. If none of the elements of one of these sets covers the instance i, by corollary 35 all the descriptions in $VS(I^+, I^-)$ do not cover the instance. Thus, the instance i is negative and the algorithm returns "−". Otherwise, the algorithm returns "?".

Instance-Classification **Algorithm**
Input: i: an instance to be classified.
$\langle I^+, \{G(I^+, \{n\})\}_{n \in I^-} \rangle$: IBMBS of $VS(I^+, I^-)$.
Output: "+" if $(\forall c \in VS(I^+, I^-))M(c, i)$.
"−" if $(\forall c \in VS(I^+, I^-))\neg M(c, i)$.
"?" otherwise.
Precondition: *Lc* is admissible, the property IP holds, and $VS(I^+, I^-) \neq \emptyset$.

label i as negative
$\langle \{I^+, \{G(I^+, \{n\})\}_{n \in I^- \cup \{i\}} \rangle$ = *Instance-Addition*$(i, \langle \{I^+, \{G(I^+, \{n\})\}_{n \in I^-} \rangle)$
if *VS-Collapse*$(\langle I^+, \{G(I^+, \{n\})\}_{n \in I^- \cup \{i\}} \rangle)$ **then**
    return "+"
for $n \in I^-$ **do**
    if $(\forall g \in G(I^+, \{n\}))\neg M(g, i)$ **then**
        return "−"
return "?".

Fig. 4. The Instance-Classification Algorithm

Example 5. Let us illustrate the classification algorithm given the IBMBS from example 1. Assume that we have to classify instance $i = \langle 1,1,1,1,1,1,0,0\rangle$. In the first step the algorithm updates the IBMBS with the instance i considered as negative. The resulting IBMBS coincide with the first IBMBS from example 2 and represent a nonempty version space. In the second step the algorithm determines that all the elements of the maximal boundary sets $G(I^+, \{i_2^-\})$, $G(I^+, \{i_3^-\})$, and $G(I^+, \{i_4^-\})$ do cover the instance. Thus, the algorithm returns "?"; i.e., the instance classification cannot be determined. □

The instance classification with the IBMBS can be extended to situations when the training instances are noisy. The key idea is to use flexible matching between instances and concept descriptions. Below we describe two procedures, based on flexible matching, for positive and negative classification, respectively.

The positive classification procedure, given an instance i to be classified, first forms the maximal boundary set $G(\emptyset, \{i\})$. Then for each positive training instance $p \in I^+$ it determines the number of descriptions $g \in G(\emptyset, \{i\})$ that do not cover the instance. If at least P_p positive training instances are not covered by at least P_g descriptions $g \in G(\emptyset, \{i\})$, the instance i is classified as positive, where P_p and P_g are parameters of flexible matching.

The negative classification procedure is similar to that given in [10]. Given an instance i to be classified, it determines for each negative training instance $n \in I^-$ the number of descriptions $g \in G(I^+, \{n\})$ that do not cover the instance i. If there exist at least N_n maximal boundary sets $G(I^+, \{n\})$ of which at least N_g descriptions do not cover the instance i, then the instance is classified as negative, where N_n and N_g are parameters of flexible matching.

5 Analysis

This section analyses the IBMBS. Subsection 5.1 gives a worst-case complexity analysis of the IBMBS and the algorithms presented. Subsection 5.2 uses the results of the analysis to determine (1) whether the IBMBS algorithms are efficient, and (2) whether the IBMBS are efficiently computable.

5.1 The Worst-Case Complexity Analysis

The worst-case complexity analysis is made in terms of the computational characteristics of admissible concept languages. The characteristics are chosen so that they do not depend on the size of the training data. They are given below:

Γ_n**:** the maximal size of the maximal boundary set $G(\emptyset, \{n\})$ for all $n \in I$;
$t_n^{\uparrow}$**:** the maximal time for generating the set $G(\emptyset, \{n\})$ for all $n \in I$;
Σ_p**:** the maximal size of the minimal boundary set $S(\{p\}, \emptyset)$ for all $p \in I$;
$t_p^{\downarrow}$**:** the maximal time for generating the set $S(\{p\}, \emptyset)$ for all $p \in I$ [2];
t_m**:** the maximal time of the operator of the relation $M(c, i)$ for all $c \in Lc, i \in I$.

[2] The computational characteristics Σ_p and $t_p^{\downarrow}$ are given for the complexity analysis of the instance-based minimal boundary sets presented in section 7.

The condition for the worst-case complexity analysis is that the size of the maximal boundary sets $G(I^+, \{n\})$ is equal to the size Γ_n for all $n \in I, I^+ \subseteq I$.

Space Complexity
The worst-case space complexity of the IBMBS is $|I^+|$ plus the worst-case space complexity $O(|I^-|\Gamma_n)$ of the G-part. Thus, it is $O(|I^+| + |I^-|\Gamma_n)$.

Time Complexities
The Instance-Addition Algorithm. The worst-case time complexity of the algorithm part for processing one positive instance is $O(|I^-|\Gamma_n t_m)$. The factor $|I^-|$ arises because we have $|I^-|$ maximal boundary sets $G(I^+ \cup \{i\}, \{n\})$. The factor $\Gamma_n t_m$ arises because in order to form each maximal boundary set $G(I^+ \cup \{i\}, \{n\})$ we test Γ_n elements of the set $G(I^+, \{n\})$ whether they cover the instance.

The worst-case time complexity of the algorithm part for processing one negative instance is $O(t_n^{\uparrow} + |I^+|\Gamma_n t_m)$. The term $O(t_n^{\uparrow})$ arises because the maximal boundary set $G(\emptyset, \{i\})$ is generated. The term $O(|I^+|\Gamma_n t_m)$ arises because the maximal boundary set $G(I^+, \{i\})$ is generated from Γ_n elements of the set $G(\emptyset, \{i\})$ that are tested to cover all the positive instances in the set I^+.

The Instance-Retraction Algorithm. The worst-case time complexity of the algorithm part for processing one positive instance is the sum $O(t_n^{\uparrow} + |I^+|\Gamma_n t_m) + O(|I^-|\Gamma_n t_m)$. The term $O(t_n^{\uparrow} + |I^+|\Gamma_n t_m)$ is the worst-case time complexity for generating the maximal boundary set $G(I^+ \setminus \{i\}, \{i\})$. (The sub-term $t_n^{\uparrow}$ arises because the set $G(\emptyset, \{i\})$ is generated. The sub-term $|I^+|\Gamma_n t_m$ arises because the size of the set $G(\emptyset, \{i\})$ is Γ_n and each element of $G(\emptyset, \{i\})$ is checked whether it covers all the positive instances in the set $I^+ \setminus \{i\}$.) The second term $O(|I^-|\Gamma_n t_m)$ is the time complexity for constructing the maximal boundary sets $G(I^+ \setminus \{i\}, \{n\})$ for all $n \in I^-$. (The factor $|I^-|$ arises because we have $|I^-|$ sets $G(I^+ \setminus \{i\}, \{n\})$. The factor $\Gamma_n t_m$ arises because formation of each set $G(I^+ \cup \{i\}, \{n\})$ requires Γ_n elements of the set $G(I^+ \setminus \{i\}, \{i\})$ to be tested not to cover the corresponding instance n.) Thus, the worst-case time complexity of this part of the algorithm is $O(t_n^{\uparrow} + |I^+|\Gamma_n t_m) + O(|I^-|\Gamma_n t_m) = O(t_n^{\uparrow} + (|I^+| + |I^-|)\Gamma_n t_m)$.

The worst-case time complexity of the algorithm part for processing one negative instance is $O(1)$ because its maximal boundary set is removed only.

The Algorithm for Version-Space Collapse. The worst-case time complexity of the algorithm is $O(|I^-|)$. The term $|I^-|$ arises because in the worst case $|I^-|$ maximal boundary sets $G(I^+, \{n\})$ are checked whether they are empty.

The Instance-Classification Algorithm. The instance-classification algorithm consists of two parts. The first part is the positive instance-classification part. Its worst-case time complexity is the sum $O(t_n^{\uparrow} + |I^+|\Gamma_n t_m) + O(|I^-|)$. (The first term is the worst-case time complexity of the instance-addition algorithm given the instance i to be classified labeled as negative. The second term is the worst-case time complexity of the algorithm for version-space collapse.) The second part is the negative instance-classification part. Its worst-case time

complexity is $O(|I^-|\Gamma_n t_m)$. The factor $|I^-|$ arises because we have $|I^-|$ maximal boundary sets $G(I^+, \{n\})$. The factor $\Gamma_n t_m$ arises because the elements of each maximal boundary set $G(I^+, \{n\})$ are tested not to cover the instance i. Thus, the worst-case time complexity of the instance-classification algorithm is: $O(|I^-|)+O(t_n^{\uparrow}+|I^+|\Gamma_n t_m)+O(|I^-|)+O(|I^-|\Gamma_n t_m) = O(t_n^{\uparrow}+(|I^+|+|I^-|)\Gamma_n t_m)$.

The IBMBS complexities are summarised in table 1.

Table 1. Worst-Case Complexities of the IBMBS and their Algorithms

Space:	$O(\|I^+\| + \|I^-\|\Gamma_n)$
Time	
Instance-Addition Algorithm ($\oplus$ instance):	$O(\|I^-\|\Gamma_n t_m)$
Instance-Addition Algorithm ($\ominus$ instance):	$O(t_n^{\uparrow} + \|I^+\|\Gamma_n t_m)$
Instance-Retraction Algorithm ($\oplus$ instance):	$O(t_n^{\uparrow} + (\|I^+\| + \|I^-\|)\Gamma_n t_m)$
Instance-Retraction Algorithm ($\ominus$ instance):	$O(1)$
Version-Space Collapse Algorithm:	$O(\|I^-\|)$
Instance-Classification Algorithm:	$O(t_n^{\uparrow} + (\|I^+\| + \|I^-\|)\Gamma_n t_m)$

5.2 IBMBS and Efficiency

To determine whether the algorithms of the IBMBS are efficient we employ a rule proposed in [2]: an algorithm of a version-space representation is efficient for a concept language if the worst-case time complexity of the algorithm is polynomial in the computational features of the language and the size of the input. From the previous subsection we know that the worst-case time complexities of the IBMBS algorithms are polynomial in the computational characteristics $t_n^{\uparrow}$, Γ_n, t_m, and the sizes $|I^+|$ and $|I^-|$. In this context we note that the upper bound of the size of the input of the algorithms is the size of the IBMBS; i.e., $|I^+|$ plus $|I^-|\Gamma_n$. Thus, we conclude that the IBMBS algorithms for instance addition, instance retraction, version-space collapse and instance classification are efficient for admissible concept languages. In addition, we emphasise that the instance-retraction algorithm does not recompute the IBMBS.

To determine whether the IBMBS are efficiently computable we employ a second rule proposed in [2]: a version-space representation is efficiently computable for a concept language if in the worst case its size is polynomial in the computational features of the language and the sizes of the training sets, and the representation has an efficient instance-addition algorithm. ¿From the previous subsection we know that the worst-case space complexity of the IBMBS is polynomial in the computational characteristic Γ_n, and the sizes $|I^+|$ and $|I^-|$. Since the IBMBS instance-addition algorithm is efficient we conclude that the IBMBS are efficiently computable for admissible concept languages.

6 Usefulness of the IBMBS

This section evaluates the usefulness of the IBMBS. For this purpose we summarise the IBMBS employing the characteristics of a useful version-space repre-

sentation (definition 8). The characteristics are compactness, finiteness, efficient computability, and efficiency of the IBMBS algorithms.

We showed that the IBMBS are a correct and compact version-space representation for admissible concept languages (section 3). They are finite if the training sets are finite and the maximal boundary set $G(\emptyset, \{n\})$ is finite for all $n \in I$. The IBMBS are efficiently computable and have efficient algorithms for instance addition, instance retraction, version-space collapse and instance classification for admissible concept languages (sections 4 and 5). The only restrictions are that the instance retraction algorithm requires the property G while the algorithms for version-space collapse and instance classification require the property IP on the concept language used.

From this summary we conclude according to definition 8 that the IBMBS are a useful version-space representation for the class of admissible concept languages if the training sets are finite, the maximal boundary set $G(\emptyset, \{n\})$ is finite for all $n \in I$, and the property G as well as the property IP hold.

7 Instance-Based Minimal Boundary Sets

Instance-based minimal boundary sets (IBmBS) and their algorithms can be derived by duality from the previous sections[3]. Therefore, we refrain from providing details. The IBmBS complexities are given in table 2.

Table 2. Worst-Case Complexities of the IBmBS and their Algorithms

Space:	$O(\|I^+\|\Sigma_p + \|I^-\|)$
Time	
Instance-Addition Algorithm ($\oplus$ instance):	$O(t_p^{\downarrow} + \|I^-\|\Sigma_p t_m)$
Instance-Addition Algorithm ($\ominus$ instance):	$O(\|I^+\|\Sigma_p t_m)$
Instance-Retraction Algorithm ($\oplus$ instance):	$O(1)$
Instance-Retraction Algorithm ($\ominus$ instance):	$O(t_p^{\downarrow} + (\|I^+\| + \|I^-\|)\Sigma_p t_m)$
Version-Space Collapse Algorithm:	$O(\|I^+\|)$
Instance-Classification Algorithm:	$O(t_p^{\downarrow} + (\|I^+\| + \|I^-\|)\Sigma_p t_m)$

8 Comparison with Relevant Work

Below we compare the IBMBS and the IBmBS with the training-instance representation [3] and the instance-based boundary-set representation [11,12], i.e., with version-space representations that are efficient for instance retraction. The comparison is made using the characteristics of useful version-space representations (definition 8).

The training-instance representation is a version-space representation that consists of the sets of positive and negative training instances. By definition the

[3] We note that the dual of the property G is the property S, and the dual of the intersection-preserving property (IP) is the union-preserving property (UP) [11].

representation is compact. Obviously, the conditions for finiteness of the training-instance representation are a subset of the conditions for finiteness of the IBMBS (IBmBS). An analogous conclusion can be derived when the representations are compared with respect to efficient computability. The training-instance representation allows much more efficient algorithms for instance addition and instance retraction. This advantage comes with a price: the instance-classification algorithm determines the classification of each instance by a search in the concept language using all the training data and the instance [2]. Thus, the training-instance representation has only a theoretical value. This contrasts with the IBMBS and the IBmBS: their instance-classification algorithms are not based on search and this is one of the factors of their usefulness.

The instance-based boundary-set representation (IBBS) is a useful version-space representation that consists of a family of minimal boundary sets indexed by positive training instances and a family of maximal boundary sets indexed by negative training instances [11,12]. The representation is correct and compact for admissible concept languages. It is possible to prove that the conditions for finiteness of the IBBS are a superset of the conditions for finiteness of the IBMBS (IBmBS). In order to compare the representations in terms of efficiency we examine the IBBS worst-case complexities given in table 3. An analysis of these complexities shows that each of them is equal to the sum of the corresponding complexities of the IBMBS and IBmBS. Thus, the IBMBS and the IBmBS have two advantages: (1) they are more efficiently computable than the IBBS, and (2) the IBMBS and the IBmBS algorithms for instance addition, instance retraction, version-space collapse, and instance classification are more efficient. Moreover, the applicability of the IBBS instance-retraction algorithm is more restricted: the algorithm can be applied only if both properties S and G hold.

Table 3. Worst-Case Complexities of the IBBS and their Algorithms

Space:	$O(\|I^+\|\Sigma_p + \|I^-\|\Gamma_n)$
Time	
Instance-Addition Algorithm ($\oplus$ instance):	$O(t_p^{\downarrow} + \|I^-\|(\Sigma_p + \Gamma_n)t_m)$
Instance-Addition Algorithm ($\ominus$ instance):	$O(t_n^{\uparrow} + \|I^+\|(\Sigma_p + \Gamma_n)t_m)$
Instance-Retraction Algorithm ($\oplus$ instance):	$O(t_n^{\uparrow} + (\|I^+\| + \|I^-\|)\Gamma_n t_m)$
Instance-Retraction Algorithm ($\ominus$ instance):	$O(t_p^{\downarrow} + (\|I^+\| + \|I^-\|)\Sigma_p t_m)$
Version-Space Collapse Algorithm:	$O(\|I^+\| + \|I^-\|)$
Instance-Classification Algorithm:	$O(t_p^{\downarrow} + t_n^{\uparrow} + (\|I^+\| + \|I^-\|)(\Sigma_p + \Gamma_n)t_m)$

9 Conclusion

This chapter introduced a family of useful version-space representations called one-sided instance-based boundary sets (IBMBS and IBmBS). We showed that these representations are correct and compact for the class of admissible concept languages. This allowed us to derive the conditions for finiteness. In addition, we demonstrated that the one-sided instance-based boundary sets are efficiently

computable and have efficient algorithms for instance addition, instance retraction, version-space collapse and instance classification for the class of admissible concept languages. We compared the one-sided instance-based boundary sets with other existing version-space representations that are efficient for instance retraction. From the comparison we conclude that the one-sided instance-based boundary sets are at the moment the most efficient useful version-space representations for instance retraction. So, our research question from section 1 has been answered positively.

References

1. Haussler, D.: Quantifying Inductive Bias: AI Learning Algorithms and Valiants Learning Framework. Artificial Intelligence **36** (1988) 177–221
2. Hirsh, H.: Polynomial-Time Learning with Version Spaces. In: Proceedings of the Tenth National Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA (1992) 117–122
3. Hirsh, H., Mishra, N., Pitt, L.: Version Spaces without Boundary Sets. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA (1997) 491–496
4. Idemstam-Almquist, P.: Demand Networks: An Alternative Representation of Version Spaces. Master's Thesis, Department of Computer Science and Systems Sciences, Stockholm University, Stockholm, Sweden (1990)
5. De Raedt, L.: Query Evaluation and Optimisation for Inductive Databases using Version Spaces, In: Online Proceedings of the International Workshop on Database Technologies for Data Mining, Prague, Czech Republic (2002)
6. De Raedt, L., Jaeger, M., Lee, S., Mannila, H.: A Theory of Inductive Query Answering, In: Proceedings of the 2002 IEEE International Conference on Data Mining, IEEE Publishing, (2002) 123–128
7. Mitchell, T.: Version Spaces: An Approach to Concept Learning. Ph.D. Thesis, Electrical Engineering Department, Stanford Univeristy, Stanford, CA (1978)
8. Mitchell, T.: Machine Learning. McGraw-Hill, New York, NY (1997)
9. Sablon, G., DeRaedt, L., Bruynooghe, L.: Iterative Versionspaces. Artificial Intelligence **69** (1994) 393–410
10. Sebag, M., Rouveirol. C.: Resource-bounded Relational Reasoning: Induction and Deduction through Stochastic Matching. Machine Learning **38** (2000) 41–62
11. Smirnov, E.N.: Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets. Ph.D. Thesis, Department of Computer Science, Universiteit Maastricht, Maastricht, The Netherlands (2001)
12. Smirnov, E.N., Braspenning, P.J.: Version Space Learning with Instance-Based Boundary Sets. In: Proceedings of The Thirteenth European Conference on Artificial Intelligence. Jonh Willey and Sons, Chichester, UK (1998) 460–464
13. Smith, B.D., Rosenbloom, P.S.: Incremental Non-Backtracking Focusing: A Polynomially Bounded Generalization Algorithm for Version Spaces. In: Proceedings of the Eight National Conference on Artificial Intelligence, MIT Press, MA (1990) 848–853