Machine Learning, 14, 193–217 (1994)

A Polynomial Approach to the Constructive Induction of Structural Knowledge

JÖRG-UWE KIETZ KIETZ@GMDZI.GMD.DE
German National Research Centre for Computer Science (GMD), Institute for Applied Information Technology, Schloß Birlinghoven, D-53757 St. Augustin, Germany

KATHARINA MORIK MORIK@LS8.INFORMATIK.UNI-DORTMUND.DE
University Dortmund, Department of Computer Science, Artificial Intelligence (LS VIII), D-44221 Dortmund 50, Germany

Abstract. The representation formalism as well as the representation language is of great importance for the success of machine learning. The representation formalism should be expressive, efficient, useful, and applicable. First-order logic needs to be restricted in order to be efficient for inductive and deductive reasoning. In the field of knowledge representation, term subsumption formalisms have been developed which are efficient and expressive. In this article, a learning algorithm, KLUSTER, is described that represents concept definitions in this formalism. KLUSTER enhances the representation language if this is necessary for the discrimination of concepts. Hence, KLUSTER is a constructive induction program. KLUSTER builds the most specific generalization and a most general discrimination in polynomial time. It embeds these concept learning problems into the overall task of learning a hierarchy of concepts.

Keywords. Constructive induction, restrictions of first-order logic for learning, learning most specific generalizations

1. Introduction

Concept learning can be described as inductively forming hypotheses expressed using a hypothesis language such that they deductively cover observations expressed using an observation language. The choice of an appropriate formalism for the hypotheses is crucial for the success of learning. On the one hand, the representation formalism should be powerful enough to express at least the relations between concepts. On the other hand, it should be efficient with respect to deductive and inductive inference. Moreover, it should be easily understandable so that experts can inspect the results of learning, and it should be in the framework of standard representations so that researchers and practitioners from other fields of computer science can easily apply the learning system.

Attribute-value representations have been in the focus of interest for several years, since they are easily understandable and applicable. Algorithms for inductive and deductive reasoning in polynomial time have been investigated (e.g., learning monomials (Kearns, 1990)). The expressive power of such representations, however, is very restricted. Therefore, first-order logic has moved into the foreground. The advantages of first-order logic are its expressive power, its understandability, and its applicability in the framework of logic programming. The disadvantage is its complexity. Without restrictions, first-order logic is not efficient, either for deductive or for inductive inference. Deduction even in Horn logic

is efficiently computable only if the clauses are programmed in a programming language with a fixed evaluation strategy (e.g., Prolog). Induction (e.g., deciding whether there is a hypothesis consistent with the examples) is polynomially computable only for a minimal subset of predicate logic (Kietz, 1993). So, first-order logic has been restricted in several ways for its use in machine learning (e.g., a restricted higher-order logic (Emde et al., 1983; Wrobel, 1987; Kietz & Wrobel, 1991, Morik et al., 1993) or datalog (Ceri et al., 1990) as used by FOIL (Quinlin, 1990) or ij-determinante Horn clauses (Muggleton & Feng, 1990)).

An alternative restriction of first-order logic has been developed in the field of knowledge representation: term-subsumption formalisms or terminological logics (Brachman & Schmolze, 1985). This representation formalism has a well-defined formal semantics. It is a greatest subset of first-order logic with deduction being still efficiently computable (Donini et al., 1991). The representation of observations and concepts is easily understandable. Several concepts can be represented by their relations to each other. The formalism is easily applicable and about to become a standard in knowledge representation. However, no learning algorithms that use a term subsumption formalism had been developed until recently. KLUSTER is the first system that learns within this framework (Morik & Kietz, 1989).[1]

In this article, we first describe the term-subsumption formalism (section 2). Then we present the learning algorithm (section 3). Its evaluation with respect to related work and in terms of a theoretical assessment is given in section 4.

2. The term-subsumption formalism used by KLUSTER

Starting with KL-ONE (Brachman, 1977; Brachman & Schmolze, 1985), an increasing effort has been spent in the development of knowledge representation systems in the framework of term-subsumption formalisms (also called terminological logic or description logic), e.g., NIKL (Moser, 1983), KL-TWO (Vilain, 1885), KRYPTON (Brachman et al., 1985), CLASSIC (Borgida et al., 1989), and BACK (Luck et al., 1987; Peltason et al., 1989). Recently, these systems have been successfully applied to a number of real-world applications (cf. Peltason et al., 1991).

The representation formalism corresponds to a rather classical view of concept descriptions, where first a set of superconcepts is referenced and then distinguishing statements are made. For instance, a motorcycle is defined as a vehicle with exactly two parts that are wheels. A car is defined as a vehicle with at least three and at most four wheels. The roles of the superconcept `vehicle` are inherited by the subconcepts, which are distinguished by number restrictions on the `part-of` role with the concept `wheels` in its range. The concept representation, i.e., the hypothesis language, is called TBox. A TBox is a semilattice with defined meets. Concepts are classified within this structure according to their super-/subconcept relation. The formalism distinguishes between primitive concepts (concept $:<$ conditions), where the conditions are necessary, but not sufficient, and defined concepts (concept $:=$ conditions), where the conditions are necessary as well as sufficient.

The observations are represented in the so-called ABox. The ABox represents assertions about individual terms. These are classified with respect to their concept membership, i.e., by their link with the TBox.

The main inferences supported by term subsumption formalisms are the classification of concepts and instances into a concept hierarchy. The classification process is formalized by the subsumption relation between concepts. This subsumption goes beyond θ-subsumption in that it respects the overall concept structure. Hence, it is similar to generalized subsumption (Buntine, 1988). The subsumption provides for a partial ordering (generality) that corresponds to logic implication within the term subsumption formalism. Term subsumption formalisms offer an expressiveness in the middle of attribute-value representations and first-order logic. They enhance the quantification of first-order logic in that they allow the specification of the minimal and the maximal number of instances for existentially quantified variables. The formal properties of various implementations of term subsumption formalisms have been investigated, and work on revisions in concept structures has been put forward (Nebel, 1990).

KLUSTER uses a formalism built from a standard set of concept- and role-forming operators proposed in the literature (e.g., Nebel, 1990; Donini et al., 1991) for representing hypotheses. The syntax follows the representation of the BACK system (Peltason et al., 1989).

<Tbox>	::=	<term-proposition>*
<term-proposition>	::=	<term-restriction>
	|	<term-introduction>
<term-introduction>	::=	<concept-introduction>
	|	<role-introduction>
<concept-introduction>	::=	<concept-name> :< <concept>
	|	<concept-name> ::= <concept>
<role-introduction>	::=	<role-name> :< <role>
	|	<role-name>: = <role>
<term-restriction>	::=	**disjoint**(<concept-name>+)
<concept>	::=	<concept-ref>
	|	**anything**
	|	**nothing**
	|	**all**(<role-ref>, <concept-ref>)
	|	**atleast**(<integer>, <role-ref>)
	|	**atmost**(<integer>, <role-ref>)
<concept-ref>	::=	<concept-name>
	|	<concept-name> **and** <concept-ref>
<role>	::=	<role-ref>
	|	<role> **and** <role>
	|	**domain**(<concept-ref>)
	|	**range**(<concept-ref>)
<role-ref>	::=	<role-name>
	|	**inverse**(<role-name>)

The only difference between this syntax and those of other TBox formalisms is the restriction in building complex expressions. Only a concept name or a conjunction of concept names is allowed in **all**, **domain**, and **range** restrictions. This eases the readability of the concept definitions and helps to avoid problems with terminological cycles (Nebel, 1990). It has no effect on the complexity of the concept learning task. Only role names or the inverse of named roles are allowed in **all**, **atleast**, and **atmost** restrictions. Not allowing complex role expressions or defined roles guarantees that the basic algorithm can compute a most specific generalization in polynomial time. If, however, defined roles are needed in order to distinguish between two disjoint concepts, these roles are introduced via constructive induction. This introduction of defined roles is bounded by parameters such that only polynomially many roles are constructed. So, constructive induction is our way out of the contradiction between the two requirements: expressiveness and efficiency (see section 3.5).

The assertional formalism (ABox) is used as the observation language by KLUSTER. Within the ABox, it is expressible that an object belongs to a concept and that two objects are related by a role.

<ABox>	:=	<assertion> +
<assertion>	::=	<object-description>
		| <relation-description>
<object-description>	::=	<concept-name>(<object>)
<relation-description>	::=	<role-name>(<object>,<object>)

KLUSTER's formalism has a standard model-theoretic semantics as follows.

Let D, the domain, be any set and E a function that maps objects to elements of D, concepts to subsets of D, and roles to subsets of D × D:

$$E: \begin{cases} <\text{object}> \rightarrow D \\ <\text{concept}> \rightarrow 2^{D} \\ <\text{role}> \rightarrow 2^{D \times D} \end{cases}$$

E is an extension function of a TBox T, if and only if for all $C_i \in$ <concept>, $R_i \in$ <role>, CA $\in$ <concept-ref>, CN $\in$ <concept-name>, RA $\in$ <role-ref>:

$$\begin{aligned}
E(\textbf{anything}) &= D \\
E(\textbf{nothing}) &= \emptyset \\
E(C_1 \textbf{ and } C_2) &= E(C_1) \cap E(C_2) \\
E(\textbf{all}(RA, CA)) &= \{x \in D \mid \forall y: <x, y> \in E[RA \Rightarrow y \in E(CA)\} \\
E(\textbf{atleast}(n, RA)) &= \{x \in D \mid |\{y \in D \mid <x, y> \in E(RA)\}| \geq n\} \\
E(\textbf{atmost}(n, RA)) &= \{x \in D \mid |\{y \in D \mid <x, y> \in E(RA)\}| \leq n\} \\
E(R_1 \textbf{ and } R_2) &= E(R_1) \cap E(R_2) \\
E(\textbf{domain}(CA)) &= E(CA) \times D \\
E(\textbf{range}(CA)) &= D \times E(CA) \\
E(\textbf{inverse}(RN)) &= \{<x, y> \in D \times D \mid <y, x> \in E(RN)\}
\end{aligned}$$

A Pair $<D, E>$, where D is a domain and E is an extension function, is a model of a TBox T and an ABox A, if and only if:

$$
\begin{array}{rcl}
E(CN) & \subseteq & E(C), \text{ for all } CN :< DC \in T \\
E(CN) & = & E(C), \text{ for all } CN := C \in T \\
E(RN) & \subseteq & E(R), \text{ for all } RN :< R \in T \\
E(RN) & = & E(R), \text{ for all } RN := R \in T \\
E(CN_i) \cap E(CN_j) & = & \emptyset, \text{ for all } CN_i, CN_j \text{ with } i \neq j, \text{ and } 1 \leq i, j \leq n \text{ in} \\
 & & \text{a } \textbf{disjoint}(CN_1, CN_2, \ldots, CN_n) \in T. \\
E(t_1) = E(t_2) & \leftrightarrow & t_1 = t_2, \text{ for all objects } t_1, t_1 \text{ in A} \\
E(t) & \in & E(C(, \text{ for all } C(t) \in A \\
<E(t_1), E(t_2)> & \in & E(R), \text{ for all } R(t_1, t_2) \in A
\end{array}
$$

The syntax and the model-theoretic semantics together define a logic. Before we define the inferences performed by the terminological reasoners, let us give an example of a well-formed concept definition and its equivalent in first-order logic with equality:

motorcycle := vehicle **and** **all**(base_part, wheel) **and** **atleast**(2, base_part) **and** **atmost**(2, base_part)

$\forall x$ (motorcycle(x) $\leftrightarrow$ vehicle(x) $\wedge$ $\forall y$ (base_part(x, y) $\rightarrow$ wheel(y)) $\wedge$
$\exists y1\ y2$ (base_part(x, y1) $\wedge$ base_part(x, y2) $\wedge$ y1 $\neq$ y2 $\wedge$
$\neg \exists y3$ (base_part(x, y3) $\wedge$ y1 $\neq$ y3 $\wedge$ y2 $\neq$ y3)))

Now, let us precisely define what we mean by subsumption, equivalence, disjointness, and incoherence of terms within a TBox T, by entailment of assertions from a TBox T and an ABox A and inconsistency of a TBox T and an ABox A.

- Within a TBox T a term t *is subsumed by* a term t′, written $t \leq_T t'$, iff for every model $<D, E>$ of T it holds that $E(t) \subseteq E(t')$.
- Within a TBox T two terms t and t′ are *equivalent*, written $t \approx_T t'$, iff for every model $<D, E>$ of T it holds that $E(t) = E(t')$.
- Within a TBox T two terms t and t′ are *disjoint*, iff for every model $<D, E>$ of T it holds that $E(t) \cap E(t') = \emptyset$.
- Within a TBox T a term t is *incoherent*, iff for every model $<D, E>$ of T it holds that $E(t) = \emptyset$.
- An assertion f is entailed by a TBox T and an ABox A, written $A \models_T f$, iff for every model $<D, E>$ of T and A it holds that $E(t) \in E(C)$ if $f = C(t)$, or $<E(t_1), E(t_2)> \in E(R)$, if $f = R(t_1, t_2)$.
- A TBox T and an ABox A are *inconsistent*, iff there exists no model $<D, E>$ of T and A.

Note, that subsumption as defined above is a semantic relation like implication or generalized subsumption (Buntine, 1988), which takes into account background knowledge. It is not a pure syntactic relation like θ-subsumption (Plotkin, 1970).

In our TBox formalism we can compute the disjointness and incoherence using subsumption or equivalance alone:

- $t \approx_T t'$, iff $t \leq_T t'$ and $t' \leq_T t$
- t is *incoherent,* iff $t \leq_T$ **nothing**
- t and t′ are *disjoint,* iff (t **and** t′) $\leq_T$ **nothing**
- $t \leq_T t'$, iff $t \approx_T$ (t **and** t′)

It is known (Donini et al., 1991) that subsumption between two concepts with respect to a TBox T in the formalism above can be decided in polynomial time, if T does not contain any role introductions and all disjoint restrictions contain only names of primitive concepts (concept names introduced by <concept-name> :< <concept>). It is also known that the formalism cannot be extended without losing the polynomial time decidability or completeness.[2] Thus, the learning result of KLUSTER cannot be classified completely polynomially if constructive induction has introduced new roles.

3. KLUSTER

In this section, we present the system KLUSTER, an inductive learning system for constructing a concept structure in the term subsumption formalism presented in the last section. A deductive reasoning system (e.g., BACK, CLASSIC) for this term subsumption formalism is assumed to be given.

The overall learning task of KLUSTER is as follows:

Given: a set of assertions in the ABox (the examples), and an empty TBox. If a partially filled TBox (the background knowledge) is given, the assertions are assumed to be saturated by entailment. Clearly, ABox and background knowledge must be consistent.

Goal: A TBox, i.e., a hierarchy of concept definitions, organizing the factual knowledge such that the concept definitions of the TBox are true in the minimal model of the ABox. The TBox can be used for inferring by entailment further descriptions about objects newly entered into the ABox.

We will use a domain of side effects of drugs for illustrating our approach. The following set of assertions is given as input to KLUSTER:

```
contains(aspirin,asa)
contains(adumbran,coffein)
contains(adumbran,oxazepun)
contains(anxiolit,oxazepun)
contains(anxiolit,finalin)
contains(adolorin,phenazetin)
contains(adolorin,prophymazon)
contains(adolorin,nhc)
contains(placo,nhc)
contains(placo,sugar)
affects(asa,headache)
affects(oxazepun, stress)
affects(finalin,stress)
contains(alka-seltzer,asa)
affects(prophymazon,headach)
affects(phenazetin,headache)
placebo(placo)
combidrug(anxiolit)
combidrug(adolorin)
monodrug(aspirin)
monodrug(alka-seltzer)
monodrug(adumbran)
anodyne(aspirin)
anodyne(alka-seltzer)
anodyne(adolorin)
sedative(anxiolit)
contains(alka-seltzer,nch)
sedative(adumbran)
active(asa)
active(finalin)
active(prophymazon)
active(phenazetin)
active(oxazepun)
add_odd(nhc)
add_on(coffein)
add_on(sugar)
excitement(stress)
pain(headache)
```

These are the given observations. No background knowledge is provided. Note, that the relation `contains` is an n-to-m relation. The first step of KLUSTER is to compute a basic taxonomy, which is a hierarchy of primitive concepts and roles based on set inclusion between the known extensions of concepts and roles. The computed basic taxonomy is used for structuring the overall task of KLUSTER into a set of concept-learning problems. The concepts that KLUSTER tries to define are taken top-down and breadth-first from the basic taxonomy. This search strategy is implemented by an agenda of concept-learning problems. Each agenda entry is a cluster of concepts (called MDC, mutually disjoint concepts) that have the same superconcept and that are mutually disjoint. This enables KLUSTER to define concepts not in isolation, but in the context in which they occur.

A concept learning problem of KLUSTER is to build discriminating definitions of the concepts of an MDC. A definition is discriminating if the number of misclassified examples is lower or equal than a given threshold ($F_{MDC} \leq \epsilon$). To test if such a discriminating definition exists, KLUSTER first builds *most specific generalizations* (MSGs) for all examples of a concept. If the available concepts and roles are not sufficient for a discriminating characterization, the representation language is expanded. This means that more complex expressions are only built if simpler ones are not sufficient. The introduction of new concepts and roles is bounded by two parameters (rlength and refinement; see section 3.5). Since the concept learning goal is to find discriminating concept definitions for the concepts of an MDC, the best (most predictive) definition is the *most general discrimination* (MGD). Therefore, KLUSTER generalizes all discriminating MSGs to MGDs. This two-step approach of learning concepts is preferred to learning MGDs directly, since the MSGs have some useful properties that MGDs do not have:

- The MSG is unique in our formalism and simple to build (see section 3.3.1).
- If the MSG is not discriminating, then no concept expression covering all positive examples is discriminating.
- The MSG is useful for a possible extension of KLUSTER to incremental learning as $msg(\{o_1, o_2, \ldots, o_n\}) = msg(\ldots msg(o_1, o_2) \ldots, o_n)$.

In our example, the following concept definitions (MGDs) are learned (see figure 1):

```
active     := substance and atleast(1, affects)
add_on     := substance and atmost(0, affects)
placebo    := drug and atmost(0, contains_active)
monodrug   := drug and atleast(1, contains_active)
              and atmost(1, contains_active)
combidrug  := drug and atleast(2, contains_active)
anodyne    := drug and all(contains_active, active_1)
              and atleast(1, contains_active)
sedative   := drug and all(contains_active, active_2)
              and atleast(1, contains_active)
```

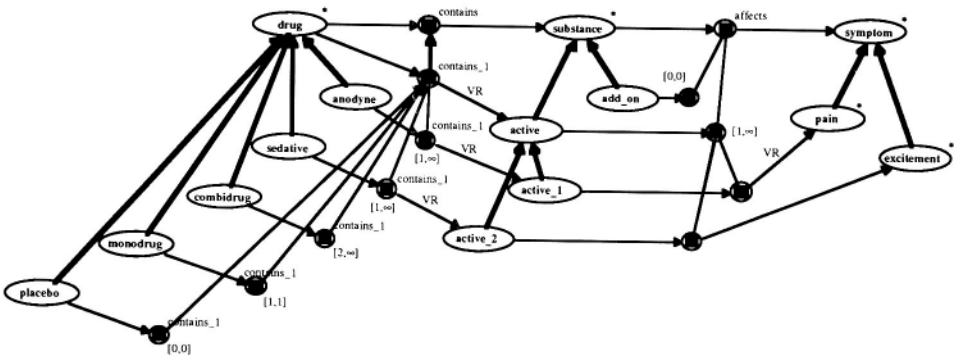

Fig. 1. The learned taxonomy for our example.

The above definitions use the following defined concepts and roles, which are introduced by KLUSTER's constructive induction:

```
contains_active := contains and range(active)
active_1        := active and all(affects, pain)
active_2        := active and all(affects, excitement)
```

The overall method of KLUSTER is summarized in table 1.

In section 3.1, we show how KLUSTER aggregates objects into primitive concepts and how the basic taxonomy of these primitive concepts is built. In section 3.2, we describe the computation of MDCs and the agenda mechanism. MSGs and the evaluation functions are defined in section 3.3. Section 3.4 presents the generalization from characterizations (MSGs) to definitions of concepts (MGDs). The constructive induction of new concepts and relations for defining a concept is described in section 3.5.

Table 1. An outline of the learning algorithm.

```
learn_TBox(ϵ, maxrefinement, maxrlength):
  begin
    compute_basic_taxonomy                                          ; building the basic taxonomy
    initialize_agenda
    repeat
      select_best_active_MDC(mdc, refinement, rlength)
      for all c ∈ mdc
        compute_and_store_MSG(c)                                    ;building MSGs
      if FMDC(mdc) ≤ ϵ                                              ;evaluating MDC
      then  set_definable_MDC(mdc)
      else  if refinement > maxrefinement ∧ rlength ≥ maxrlength
            then  set_undefinable_MDC(mdc)
            else  build_refinements(mdc, refinement, rlength)       ;constructive induction of concepts, roles
    until all mdc ∈ agenda; definable_MDC(mdc) ∨ undefinable_MDC(mdc)
    for all definable_MDC(mdc)
      for all c ∈ mdc
        compute_and_store_MSG(c)                                    ;building MSGs with enhanced language
        generalize_MSG_to_MGD(c)                                    ;building MGDs
    delete_all_refinements_not_used_in_MDGs
  end
```

3.1. Building the basic taxonomy

As the first step of learning, KLUSTER aggregates objects of the ABox into primitive concepts of the TBox. Objects that occur in the ABox as an argument of a one-place predicate are collected as the known extension of a primitive concept in the TBox named by the predicate symbol. Tuples of objects, which occur in a two-place predicate of the ABox, are interpreted as the known extension of a primitive role in the TBox named by the predicate symbol. The domains and ranges of the primitive roles are also determined. The domain of a role is the set of objects occurring at the first place of the role. The range of a role is the set of objects occurring at the second place of the roles.

Let us describe this more formally. Let ext be an extension function as defined in section 2:

$$\text{ext:} \begin{cases} <\text{object}> \rightarrow <\text{object}> \\ <\text{concept}> \rightarrow 2^{<\text{object}>} \\ <\text{role}> \rightarrow 2^{<\text{object}> \times <\text{object}>} \end{cases}$$

where $\text{ext}_{<\text{object}>}$ is the identity function between the objects in the ABox, i.e., the objects of the ABox are the domain of the interpretation. Then the pair $<<\text{object}>,\text{ext}>$ is a minimal model of the given TBox and ABox, if TBox and ABox are consistent and the ABox is complete with respect to the given TBox. This is always the case if the TBox is empty, i.e., if no background knowledge is given.

The system then builds root concepts as the union of all extentionally overlapping domains and ranges of roles and primitive concepts. The root concepts are similar to the sorts or types that other learning systems (e.g., FOIL (Quinlan, 1990); GOLEM (Muggleton & Feng, 1990) take as input.

Then the primitive concepts are arranged into a hierarchy based on set inclusion of the extensions. This means that the subsumption relationships valid in the minimal model $<<\text{object}>,\text{ext}>$ are induced. Since subsumption is a partial ordering, a minimal representation consists of the direct subsumptions. KLUSTER uses standard algorithms to compute the direct subsumption from subsumption.

Disjointness of primitive concepts is also determined based on the extensions, i.e., all disjoint relationships valid in the minimal model $<<\text{object}>,\text{ext}>$ are induced. As disjoint restrictions are inherited along subsumption, the system computes the minimal set of disjoint restrictions necessary to infer the inherited ones. The disjointness of `anodyne` and `pain`, for example, can be inferred from the disjointness of `drug` and `symptom`, since `drug` subsumes `anodyne` and `symptom` subsumes `pain`.

In our example, root concepts (the predecessor of **anything**) and primitive concepts are:[3]

```
drug       :< anything,  ext(drug) = {adolorin,adumbran,alka_seltzer,anxiolit,aspirin,placo}
placebo    :< drug,      ext(placebo) = {placo},
monodrug   :< drug,      ext(monodrug) = {adumbran,alka_seltzer,aspirin},
combidrug  :< drug,      ext(drug) = {adolorin,anxiolit},
anodyne    :< drug,      ext(anodyne) = {adolorin,alka_seltzer,aspirin},
sedative   :< drug,      ext(sedative) = {adumbran,anxiolit},
substance  :< anything,  ext(substance) = {asa,coffein,finalin,nhc,oxazepun,phenazetin,prophymazon,sugar}
active     :< subtance,  ext(active) = {asa,finalin,oxazepun,phenazetin,prophymazon},
add_on     :< substance, ext(add_on) = {coffein,nhc,sugar},
symptom    :< anything,  ext(symptom) = {bellyache,headache,stress},
pain       :< symptom,   ext(pain) = {bellyache,headache},
excitement :< symptom,   ext(excitement) = {stress}.
```

All concepts are primitive, i.e., they still need to be defined. The minimal set of disjoint restrictions is the following:

```
disjoint(drug, substance),     disjoint(drug, symptom),       disjointsubstance, symptom),
disjoint(placebo, monodrug),   disjoint(placebo, combidrug),  disjoint(monodrug, combidrug),
disjoint(placebo, anodyne),    disjoint(placebo, sedative),   disjoint(anodyne, sedative),
disjoint(active, add_on)       disjoint(pain, stress)
```

The roles of the basic taxonomy are

```
contains     :< domain(drug) and range(substance), with
ext(contains)={(adolorin, nhc), (adolorin, phenazetin), (adolorin, prophymazon), (adumbran, coffein),
               (adumbran, oxazepun), (alka_seltzer, asa), (alka_seltzer, nhc), (anxiolit, finalin),
               (anxiolit, oxazepun), (aspirin, asa), (placo, nhc), (placo, sugar)}
affects      :< domain(active) and range(symptom), with
ext(affects)={(asa, headache), (finalin, stress), (oxazepun, stress), (phenazetin, bellyache), (prophymazon, headache)}
```

Figure 2 shows the basic taxonomy that is the result of the first step of KLUSTER for our example.

3.2. The concept learning problems of KLUSTER

Having computed the basic taxonomy, KLUSTER sets up concept learning problems. The concept learning goal is to define primitive concepts preserving the discrimination from their sister concepts. Sister concepts are the mutually disjoint subsconcepts of a common superconcept. They are called *mutually disjoint concepts* (MDCs). There can be more than one MDC for a superconcept. This is the case if a concept can be specialized with respect to diverse aspects. For instance, in our example domain, drugs are classified with respect to the combinations of substances into `monodrugs`, which consist of only one effective substance, `combidrugs`, which consist of more than one effective substance, and `placebos`, which consist of no effective substance. These three primitive concepts together

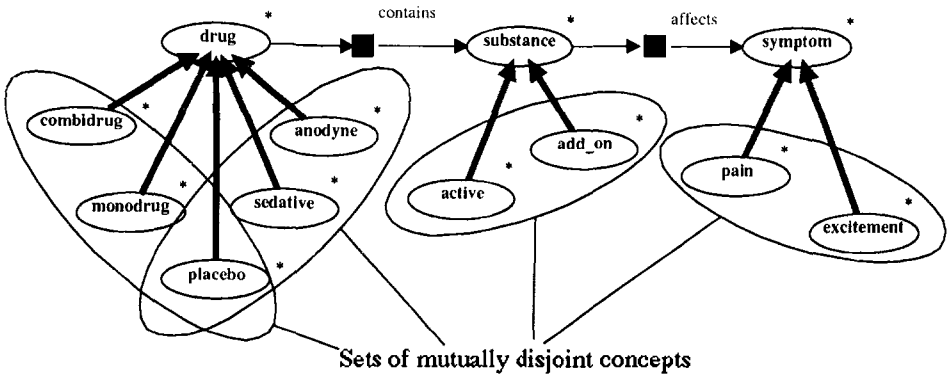

Fig. 2. The basic taxonomy for our example.

form an MDC. According to the effect they have on the human body, drugs are also classified into painkillers (anodyne), stress removers (sedative), and placebos (no effect at all). These primitive concepts together form another MDC. Both classifications are appropriate, i.e., a cross-classification of drugs is desired. Both MDCs are to be defined. To define concepts of an MDC such that no instance is covered by more than one concept of an MDC is the concept learning problem of KLUSTER.

In order to set up concept learning problems, MDCs are first built for each root concept. Pairwise disjointness is already computed. Mutual disjointness of several primitive concepts is computed by first establishing the complementary list of nondisjoint pairs. Then a list of all primitive concepts that occur in any of the disjoint pairs is split according to the nondisjoint pairs. The list of nondisjoint pairs is checked exhaustively. The lists resulting from splitting are the MDCs. Computing these maximal sets of mutually disjoint concepts is computational expensive in the worst case. The computational costs for m MDC is $m \log_2 m$. However, for n concepts, there are in the worst case $\binom{n}{n/2}$ different MDCs.[4]

MDCs are then ordered on an agenda. The agenda determines a top-down, breadth-first order of concepts to be defined. In terms of the graphic representation of the concept structure, MDCs are set up as concept learning problems from left to right, one level at a time. In our example, there are two MDCs for drugs, one for substances, and one for symptoms at the beginning:

```
MDC_1: {placebo, monodrug, combidrug}
MDC_2: {placebo, anodyne, sedative}
MDC_3: {active, add_on}
MDC_4: {pain, excitement}
```

Since root concepts are not defined, the concept structure will never consist completely of defined concepts. Most often, not all of the concept learning tasks are accomplished by KLUSTER. Some concepts remain primitive. They assist the definition of other concepts without being defined themselves. The reason for this is that we want to prohibit circular definitions. Hence, there are always concepts that are used in **all** restrictions of concept definition but are not defined themselves. In the graphic representation of our example, these are the rightmost concepts. Only if the rightmost concepts are definable using number restrictions can they be defined while still avoiding the pitfall of circular definitions.

3.3. Characterizing concepts

The definition of concepts is performed in three steps by KLUSTER. First, concepts are characterized by the *most specific generalization* (MSG). Then the characterization is evaluated. Finally, the characterization is further generalized to become the *most general discrimination* (MGD), which is the definition of the concept. A characterization selects all relevant roles, whereas a definition selects the most discriminating ones among the relevant roles. We first describe the induction resulting in an MSG. Then we describe the evaluation of all concept characterizations of an MDC. Finally (section 3.4), we describe how the acceptable MSGs are further generalized to an MGD.

3.3.1. Building the most specific generalizations (MSG)

The MSG of a concept c is a set of most specific concept expression ce, such that c :< ce is true in the minimal model <<object>, ext>, or more formally:

$$\mathrm{MSG}(c) = \{\, ce \in \langle \mathrm{concept} \rangle \mid \mathrm{ext}(c) \subseteq \mathrm{ext}(ce) \wedge \neg\, \exists\, ce_2 \in \langle \mathrm{concept} \rangle : \mathrm{ext}(c) \subseteq \mathrm{ext}(ce_2) \subset \mathrm{ext}(ce) \,\}$$

In our term subsumption formalism, there are several semantically equivalent concept expressions, that are not syntactically identical. Fortunately, there exists a unique normalized concept expression for every equivalence class in our term subsumption formalism. To determine this unique normalized expression, let us look at the possible concept expressions in our formalism. A concept expression is a conjunction (**and**) of concept names, **all**-, **atleast**-, and **atmost**-restrictions. Clearly, the **and** operator is commutative, associative, and idempotent. This means that the order of the restrictions is irrelevant. Now, let us look at the particular restrictions possible in concept expressions and their normalization:

- Concept names: these are the superconcepts of the concept. Normalization selects the direct predecessors of the concept in the subsumption partial ordering.
- **all**-restrictions: these are the expressions of the form: **all**(r, c_1 **and** . . .**and** c_n), where r is a role name or the inverse of a named role, and the c_i are concept names. Two **all**-restrictions with the same role r in a concept expression, **all**(r, c_1 **and** . . .**and** c_n) **and** **all**(r, c_{n+1} **and** . . .**and** c_{n+m}), are normalized to the equivalent expression, **all**(r, c_1 **and** . . .**and** c_n **and** c_{n+1} **and** . . .**and** c_{n+m}). The c_1 **and** . . .**and** c_n in **all** restrictions are also normalized based on the subsumption partial ordering. If the range of a role r is C, an **all**(r, C) restriction is equivalent to **anything** and can be dropped.
- **atleast**-restrictions: these are the expressions of the form **atleast**(l, r), where r is a role name or the inverse of a named role. Two **atleast**-restrictions with the same role r, namely, **atleast**(l_1, r) **and** **atleast**(l_2, r), are normalized to the equivalent expression **atleast**(maximum(l_1, l_2), r). An **atleast**(0, r) restriction is equivalent to **anything** and can be dropped.
- **atmost**-restrictions: these are the expressions of the form **atmost**(m, r), where r is a role name or the inverse of a named role. Two **atmost**-restrictions with the same role r, namely, **atmost**(m_1, r) **and** **atmost**(m_2, r), are normalized to the equivalent expression **atmost**(minimum(m_1, m_2), r). An **atmost**(0, r) restriction can be dropped if the domain of r and a superconcept of the concept under normalization are necessarily disjoint.

It is clear that any concept expression normalized as above contains for every role name r at most one **all**-restriction, at most one **atleast**-restriction and at most one **atmost**-restriction for r and for **inverse**(r). This means that the size of any concept expression in our formalism is polynomially bound in the number of concept names and role names and the coding size of the greatest integer used in the TBox.

Another important consequence of the normalization is that there are always at most finitely many concept expressions not within subsumption order. By definition, the MSG

contains only expressions that are not in a subsumption relation. So the MSG contains at most finitely many concept expressions. Suppose that there would be two different concept expressions within the MSG. This implies also that the conjunction of both is a generalization of all examples. Clearly, the conjunction of two concepts is more specialized than the concepts it is built from. This implies that both are not MSGs but that the conjunction is. Since the conjunction can be built, it will be the MSG. This proves that the MSG of a concept is unique (under equivalence) in our term subsumption formalism.

From this it becomes clear how the unique, normalized MSG of a concept c is constructed:

The superconcepts of c are already computed in the basic taxonomy.
For each role name r,
  If the domain of r is not disjoint to a superconcept of c add
    all(r, c_1 **and** ...**and** c_n), for all smallest c_i,
        which fulfill $\{ y \mid (x, y) \in r \wedge x \in ext(c) \} \subseteq ext(c_i)$
    atleast(l, r), where l = minimum($|\{ y \mid (x, y) \in est(r) \}|$, for all $x \in ext(c)$)
    atmost(m, r), where m = maximum($|\{ y \mid (x, y) \in est(r) \}|$, for all $x \in ext(c)$)
    to the MSG of c.
  If the range of r is not disjoint to a superconcept of c add
    all(**inverse**(r), c_1 **and** ...**and** c_n), for all smallest c_i,
    which fulfill $\{ y \mid (y, x) \in r \wedge x \in ext(c) \} \subseteq ext(c_i)$
    atleast(l, **inverse**(r)), where l = minimum($|\{ y \mid (y, x) \in ext(r) \}|$, for all $x \in ext(c)$)
    atmost(m, **inverse**(r)), where m = maximum($|\{ y \mid (y, x) \in ext(r) \}|$,
                for all $x \in ext(c)$)
    to the MSG of c.

In the example, the characterizations for the concepts are

```
MSG(placebo) = drug and all(contains, add_on) and atleast(2, contains) and atmost(2, contains)
MSG(monodrug) = drug and atleast(1, contains) and atmost(2, contains)
MSG(combidrug) = drug and atleast(2, contains) and atmost(3, contains)
MSG(sedative) = drug and atleast(2, contains) and atmost(2, contains)
MSG(anodyne) = drug and atleast(1, contains) and atmost(3, contains)
MSG(active) = substance and atleast(1, affects) and atmost(1, affects) and
              atleast(1, inverse(contains)) and atmost(2,inverse(contains))
MSG(add_on) = substance and atmost(0, affects)
              atleast(1, inverse(contains)) and atmost(1, inverse(contains))
MSG(pain) = symptom and atleast(1, inverse(affects)) and atmost(2, inverse(affects))
MSG(excitement) = symptom and atleast(2, inverse(affects)) and atmost(2, inverse(affects))
```

3.3.2. Evaluating MSGs in context

The evaluation of most specific generalizations is performed in the context of the MDC. The purpose of the evaluation is to accept or reject a concept characterization. As opposed to decision tree induction or conceptual clustering, the evaluation is *not* concerned with the selection of the best characterization among other alternatives because there is always

exactly one MSG for a concept. If an MSG does not get a perfect evaluation, we know that by using the given representational entities there can be no acceptable generalization. This is the criterion for introducing new concepts or relations (see section 3.5).

The following points are evaluated:

- how well the overall MDC is characterized, i.e., the MDC failure,
- how well an MSG separates a concept from the other concepts of the same MDC, i.e., the MSG failure,
- how much the restrictions of a particular role within the MSGs of the concepts of the MDC contributes to the separation within an MDC, i.e., the role failure,
- how well the restrictions of a particular role within the MSG describe a concept, i.e., the restrictions failure.

The overall evaluation of characterizations of an MDC, the MDC failure, is simply the sum of all MSG failures divided by the number of concepts of the MDC. The MSG failure counts how many instances of an MSG are also instances of other concepts of the MDC and normalizes this number by dividing it by the number of objects of the MDC. The formula for the MSG failure is:

$$F_{MSG}(c, mdc) = \frac{|ext(MSG(c)) \cap (\bigcup_{mc \in mdc,\, c \neq mc}^{mc} ext(mc)|}{|\bigcup_{mc \neq mdc}^{mc} ext(mc)|}$$

The role failure measures the contribution of a role R to the discrimination of the concepts of an MDC. It sums up all restrictions failures for a role and normalizes this by dividing the sum by the number of objects of the MDC. Hence, the basis for the role failure is the restrictions failure. The restrictions failure counts how may objects of another concept of the MDC are covered by the restrictions in the MSG using only a particular role. This count is then divided by the number of objects of the MDC. We use this for formalizing the restriction failure:

$$F_R(r, c, mdc) = \frac{|ext(rr(r, c)) \cap (\bigcup_{mc \in mdc,\, c \neq mc}^{mc} ext(mc)|}{|\bigcup_{mc \neq mdc}^{mc} ext(mc)|}$$

where rr(r, c) := **all**(r, vc) **and atleast**(l, r) **and atmost**(m, r) within the MSG(c).

In the example, for MDC_1 the extensions of placebo, monodrug, and combidrug according to their characterization are

ext (rr(contains, placebo)) = ext(MSG(placebo)) = {placo}
ext (rr(contains, monodrug)) = ext(MSG(monodrug)) = {alka-seltzer, aspirin, adumbran, placo}
ext (rr(contains, combidrug)) = ext(MSG(combidrug)) = {odolorin, anxiolit, alka-seltzer, adumbran, placo}

The underlined objects are intersections with other concepts of MDC_1. They are misclassified by the characterization. The restrictions failures are

F_R (contains, placebo, MDC_1) $= F_{MSG}$(placebo, MDC_1) = 0
F_R (contains, monodrug, MDC_1) $= F_{MSG}$(monodrug, MDC_1) = 1/6
F_R (contains, combidrug, MDC_1) $= F_{MSG}$(combidrug, MDC_1) = 3/6

The role failure for contains is the sum of the restrictions failures divided by the number of concepts of MDC_1. This is equal to the overall MDC failure since contains is the only role involved:

F_{RMDC} (contains, MDC_1) = F_{MDC} (MDC_1) = 4/18

For MDC_2, the extensions of placebo, sedative, and anodyne according to their characterization are

ext (rr(contain, placebo)) = ext(MSG(placebo)) = {placo}
ext(rr(contains, sedative)) = ext(MSG(sedative)) = {adumbran, anxiolit, alka-seltzer, placo}
ext(rr(contains, anodyne)) = ext(MSG(anodyne)) = {adolorin, alka-seltzer, aspirin, adumbran, anxiolit, placo}

The restrictions failures are

F_R (contains, placebo, MDC_1) $= F_{MSG}$(placebo, MDC_1) = 0
F_R (contains, sedative, MDC_1) $= F_{MSG}$(sedative, MDC_1) = 2/6
F_R (contains, anodyne, MDC_1) $= F_{MSG}$(anodyne, MDC_1) = 3/6

The role failure for contains, as well as the over MDC failure of MDC_2, is

F_{RMDC} (contains, MDC_1) = F_{MDC} (MDC_1) = 5/18

These failures are rather high, which shows that the characterizations are not specific enough. But the most specific generalizations have already been built; using the concepts and roles given, there are no more specific characterizations. Thus, either new concepts and/or roles must be built that can contribute to a better discrimination, or the MDCs must be marked as undefinable and taken away from the agenda of concept learning tasks. When describing the introduction of new concepts and relations (section 3.5), we shall come back to these examples.

MDC_1 uses only one role for its concepts. Thus, the role failure is the same as the MDC failure. MDC_2 is more interesting since there are two roles involved in the characterizations of each concept:

```
ext(RR (affects, active))            = {phenacetin, asa, prophymacon, oxacepun, finalin}
ext(RR (inverse (contains, active))  = {phenacetin, asa, prophymacon, oxacepun, finalin, sugar, coffein, nhc}
ext(RR (affects, add_on))            = {sugar, coffein, nhc}
ext(RR (inverse (contains), add_on)) = {phenacetin, asa, prophymacon, oxacepun, finalin, sugar, coffein, nhc}
```

Using the role affects gives no misclassification for active nor for add_on. The restrictions failure is 0 in both cases. Characterizing the concepts by the inverse of the role contains, however, makes for the restrictions failure of 3/8 for active and 5/8 for add_on. The overall role failure is 0 for affects and 1/2 for the inverse of contains. Since the MSG failure measures the failure from the conjunction of the restrictions for each concept, it is 0, too, for both active and add_on. Therefore, the MDC failure is also 0. From the comparison of the role failure with the MDC failure, it becomes clear that the concepts can be defined without using the relation inverse(contains):

F_{RMDC} (inverse(contains), MDC_2) = 1/2, but
F_{MDC} (MDC_2) = 0

This information will be used in the shift from characterizations to definitions.

3.4. The shift from characterizations to definitions or building the MDG

Definitions of concepts are intended to cover more than the observed objects, but not objects that are classified into a disjoint concept. Definitions are supposed to be as short as possible, and they should all use the same roles, if possible. Finally, they should not be cyclic.

The generalization of MSGs to MGDs is performed by dropping and generalizing restrictions as long as the discrimination is preserved. In our example, the MGDs for active and add_on are

```
MGD(active) = substance and atleast(1, affects)
MGD(add_on) = substance and atmost(0, affects)
```

All the restrictions involving the inverse relation contains are dropped, and for active the **atmost**-restriction of affects is also dropped.

When no further restriction can be dropped, KLUSTER tries to generalize the restrictions. **all**-restrictions are generalized by generalizing the concept reference, i.e., replacing a concept by its superconcepts or simply dropping a conjunct. **atleast**-restrictions are genealized by decreasing the number, and **atmost**-restrictions by increasing the number. KLUSTER generalizes as long as no misclassification is introduced.

There can be several MGDs. In principle, from n relevant restrictions, m restrictions are sufficient for discrimination, i.e., in the worst case there are $\binom{n}{n/2}$ different minimal concept definitions. But KLUSTER enters the first found MGD into the concept structure, instead of looking for the best one. Therefore, no combinatorial explosion can occur. The algorithm drops a restriction and checks whether the remaining definition leads to a misclassification. If no misclassification occurs, the restriction is dropped; otherwise, it is kept. Then the next restriction is tested in a similar manner. This guarantees that a most general still discriminating generalization is achieved.

3.5. Forming new concepts and relations

As illustrated by the example of placebos, mondrugs, and combidrugs, as well as placebos, sedatives, and anodynes, sometimes a good MSG cannot be built using the given concepts and roles. However, if a role or a concept in **all**-restrictions can be specialized, the new, specialized roles or concepts can be used for a more special characterization. The MDC, which is not definable before, is marked as waiting on the agenda, and the new roles or concepts are put on the agenda.

Specialization is performed using two rules:

- If two concepts (C11 and C12 in figure 3) of an MDC have the same concepts in the **all**-restriction of a role (C2 in figure 3), but the range of the role is in fact disjoint for the two concepts, then introduce new subconcepts of the concept in the **all**-restriction and describe the **all**-restrictions in terms of new concepts.
- If the concept (C2 in figure 3) in the **all**-restriction of a role has disjoint subconcepts, introduce new relations that are restricted to these subconcepts and try them for characterization.

In the example of monodrug, combidrug, and placebos, the second rule applies. The concept substance has two disjoint subconcepts, active and add_on. The relation contains is specialized into contains_active, which relates drugs and active, and the relation contains_add_on, which relates drugs and add_on. Then, the MDC_1 is put back on the agenda as active, with the counter refinement increased by one. Based on the parameter max_refinement, at most $(|\text{ role } | * | \text{ concept } |)^{\text{max\_refinement}}$ different new roles are introduced by the second refinement rule. When MDC_1 is next selected from the agenda, these new roles are also tried for characterization. This leads to the following MSGs.:

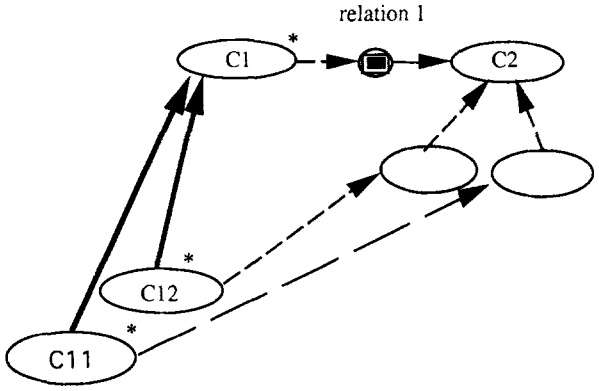

Figure 3. Illustration of the refinement rules.

```
MSG (placebo) = drug and all(contains, add_on) and atleast(2, contains) and atmost(2, contains)
                atmost(0, contains_active) and
                atleast(2, contains_add_on) and atmost(2, contains_add_on)
MSG (monodrug) = drug and atleast(1, contains) and atmost(2, contains) and
                atleast(1, contains_active) and atmost(1, contains_active) and
                atmost(1, contains_add_on)
MSG (combidrug) = drug and atleast(2, contains) and atmost(3, contains) and
                atleast(2, contains_active) and atmost(2, contains_active) and
                atmost(1, contains_add_on)
```

The evaluation results in a role failure of 0 for contains_active and of 1/3 for contains_add_on; the failure for contains remaining the same as above. Therefore, the MGDs as presented in section 3 are built using uniformly contains_active. As the definitions are entered, MDC_1 is marked as definable.

In the example of monodrug, anodyne, and sedative, now the first rule applies. The concept active used in the **all**-restriction of the role contains_active can be specialized into two disjoint subconcepts, active_1 and active_2. The following extensions are assigned:

```
ext(active_1) = {finalin, oxazepun},
ext(active_2) = {asa, phenazetin, prophymazon}.
```

These new concepts form MDC_5. This new MDC_5 is put on the agenda as active, and the counter rlength (for role chain length) is set to the one for MDC_2 plus 1. Then MDC_2 is marked on the agenda as waiting for a definition MDC_5. Based on the parameter max_rlength, at most $(|\text{ role }| * |\text{ concept }|)^{\text{max\_rlength}}$ different new MDCs can be introduced by the first refinement rule. When MDC_5 is selected from the agenda, the built MSGs are without failure, since the **all**-restriction of affects is sufficient for discrimination. Therefore MDC_5 is marked definable, and MDC_2 is set to active. The new concepts are then sufficient to discriminate the concepts of MDC_2 based on an **all**-restriction of the role contains_active (see the MGDs in section 3).

4. Evaluating KLUSTER

In the following sections, we evaluate our approach. First, we describe the theoretical properties of KLUSTER. Then we compare KLUSTER with other work on conceptual clustering, learning relational concept definitions, and constructive induction.

4.1. Theoretical evaluation

In the following, we want to evaluate KLUSTER theoretically. We first characterize the learning result of KLUSTER, and then indicate the certainty of finding the MSG for a set of facts and the time complexity of the algorithm.

KLUSTER's learning result consists of root concepts (which correspond to user-given sorts of other learning systems), several hierarchies and their interrelations, and newly constructed concepts and roles. Number restrictions are also learned. The learning result is

represented within a term subsumption formalism, which has a well-defined semantics. It provides classification with inheritance. The **all**-restrictions and the number-restrictions of the formalism are computable in polynomial time. In this respect, the term subsumption formalism goes beyond the common restrictions of first-order logics. In particular, the formalism is not restricted to ij-determinate clauses (Muggleton & Feng, 1990). It has been shown that the term subsumption formalism is one of the greatest subsets of first-order logic with decidability in polynomial time (Donini et al., 1991). Therefore, it is a promising alternative to other restrictions of first-order logics. The learning result is easily understandable because the concept structure corresponds to a classical view of concept definitions. Hybrid representation systems with a TBox in the term subsumption formalism and with facts in the ABox are becoming widely used. The KLUSTER algorithm can be incorporated into such hybrid systems in order to make them easier to use.

The restrictions of the formalism concern truly disjunctive concepts and the transitivity of relations. This includes recursive concepts that require a termination condition. So, for example, `member` cannot be learned by KLUSTER. Recursive concepts such as `ancestor` can be learned. However, many term subsumption systems do not allow recursive concepts (terminological cycles). These systems cannot fully use KLUSTER's learning result. Another restriction of the term subsumption formalism is that it cannot express transitivity where more than two variables need be bounded within the same expression. For instance, it cannot be stated directly that a drug containing a substance that increases blood pressure also increases blood pressure. In order to express this information, a new subconcept of drugs must be defined by its relation to those substances that raise blood pressure.

It is certain that KLUSTER finds an MSG for any concept. This is due to the concept representation in which exactly one MSG can be constructed for any set of terms. As was shown in section 3.3, this MSG is constructed by KLUSTER. If there exists a concept definition that is consistent with the ABox, then KLUSTER will determine it in polynomial time. If KLUSTER does not find a concept definition, then no hypothesis exists that is consistent with the minimal model of the ABox.

It may happen, however, that the failure of an MSG is greater than the threshold of MSG failure. This means that the MSG covers all positive instances but also instances of a disjoint concept. In this case, KLUSTER does not shift from the MSG to the MGD and does not enter a concept definition into the concept structure. The concept is indicated to be explored but not defined. Three cases can be distinguished. In the first, the concept cannot be expressed by a conjunction but is a truly disjunctive concept. KLUSTER cannot learn disjunctive concepts. In the second, the concept cannot be defined using the given representation language. In this case, the specialization rules may introduce new concepts or roles that then allow for definition of the concept. However, in contrast with Shapiro's refinement operator (Shapiro, 1983), KLUSTER's specialization is not complete. Therefore, in the third case, a concept is undefinable because its definition lies outside of the hypothesis space enlarged by the specialization for introducing new terms.

KLUSTER does not need very many instances for learning. KLUSTER delivers already an MSG for just one example. In this case, the MSG corresponds to the classifiction of instances as performed by term subsumption formalisms. The time for finding an MSG grows polynomially in the number of instances (and roles and concepts in the all-restriction). Therefore, KLUSTER is able to run on large example sets. The most time-consuming part

is the calculation of the MDCs. This information, however, need not be given by the user (as is the case for many other learning systems) but is acquired by KLUSTER.

KLUSTER does not require the user to build the background knowledge carefully in order to enable successful learning. Instead, KLUSTER acquires the information that is represented as background knowledge by other learning systems (e.g., DISCIPLE (Kodratoff & Tecuci, 1989)). Since the most specific generalization is exactly determined with respect to the given examples, incomplete descriptions of objects (e.g., a combidrug that contains only one active instance) prevent KLUSTER from learning the user-intended concept definition (e.g., combidrugs having more than one active substance). A user who is not content with KLUSTER's learning result may input additional facts. In this way, KLUSTER can be used as an aid in inspecting data.

Computing the basic taxonomy by KLUSTER is of polynomial complexity over the number of facts. The MDCs are computable in the average case, but in the worst case there are exponentially many different MDC. It is $m \log_2 m$ to compute m MDCs. However, for n concepts, there are at most $\binom{n}{n/2}$ different MDCs. Building the MSG is polynomial over the number of instances, the number of roles, and the number of concepts for the **all**-restriction of a role. It is polynomial because only named concepts and roles are used for **all**-restrictions. This is an incompleteness with respect to the expressability of term subsumption formalisms that allow more complex expressions. As is often the case, incompleteness makes the task solvable in polynomial time. If no named concept or role can be found for restricting a role's range, then constructive induction can define such a concept or role by specialization. The specialization step is bounded by two parameters: the depth of specialization (i.e., a specialized concept or role can be further specialized and so on, but there is a specialization that will not be further specialized) and the number of trials to define an MDC. These bounds prevent the specialization step from combinatorial explosion. Further work is planned concerning the trade-off between the formalism's expressability and the complexity of the concept learning task and on relating this to complexity results of others (e.g., Haussler, 1989). A preliminary study is that of Kietz (1992).

4.2. Related work

The learning result of KLUSTER is a concept structure that is capable of expressing cross-classifications, hierarchies for several root concepts, and the cardinality of roles. A concept structure of this type is not learned by any other learning system. Therefore, it is hard to compare KLUSTER with other systems. In the following, we compare KLUSTER with conceptual clustering algorithms because the overall task of the system is to learn a hierarchy of concepts. With respect to KLUSTER's concept learning problem, it is compared with other learning algorithms that acquire structural concept definitions. As KLUSTER introduces new terms into the hypothesis language, it is also compared with other constructive induction algorithms.

4.2.1. Conceptual clustering

The learning goal of conceptual clustering methods as well as that of KLUSTER is a hierarchy of concepts. The attribute based conceptual clustering methods, e.g., COBWEB (Fisher, 1987), UNIMEM (Lebowitz, 1987), and WITT (Hanson & Bauer, 1989), require that all instances are described along the same attributes. This approach is not suitable to describe really different—but nevertheless related—things, such as drugs, substances, and symptoms. The complete attribute vectors are also a kind of segmentation into completely described observations. Even the relational conceptual clustering system KBG (Bisson, 1990) needs a segmentation of the input into observations, and it clusters only the observations and not the objects involved in them. KLUSTER does not require such a segmentation of the input. KLUSTER does *learning from examples* instead of clustering observations. It intentionally defines sets of objects involved in examples. LABYRINTH (Thompson & Langley, 1989), another approach for relational clustering, also requires a segmentation into observations. Its main task is to cluster these observations, but it also tries to cluster the objects occurring in the observations. LABYRINTH suffers from combinatorial explosion when it tries to find an optimal mapping between the different objects involved in an observation. KLUSTER does not encounter this explosion because it uses the **all**-restriction instead of an optimal mapping of the involved objects.

4.2.2. Learning structural descriptions

KLUSTER is comparable with logical concept learning approaches such as, RLGG (Plotkin, 1970), GOLEM (Muggleton & Feng, 1990), FOIL (Quinlan, 1990) in that it learns relational concept definitions. KLUSTER requires both unary and binary relations as input. The quantity and quality of given examples is irrelevant. KLUSTER reflects the quality of the examples by the output of MSGs that cover the examples. Kietz (1992) shows that learning MSGs (RLGGs) in Horn Logic is in general intractable. GOLEM's restriction to depth-bounded determinate Horn clauses is one possible way to come to polynomial learnability. KLUSTER's MSGs with the **all**-restriction offer another possibility for polynomial learnability. The difference is that GOLEM requires all objects in the examples to be reachable by deterministic relations; hence GOLEM is not applicable to our drug example, since `contains` is a nondeterminate relation. A `drug` contains many `substances`, so `contains` is really a relation and not a function. If the substances of a drug are encoded as a list (so that `contains` becomes ij-determinate), then accessing one of the contained substances, which is necessary for defining `anodyne` and `sedative`, requires the nondeterminate `member` relation. In contrast, KLUSTER allows nondeterministic relations (e.g., `contains` in the examples). The nondeterminate relations in the examples are abstracted into one expression (the **all**-restriction) describing the similarities of all related objects.

The heuristic learning approach FOIL is also capable of using nondeterminate relations. Running FOIL on our side effect of drug data gave the following results:[5]

```
anodyne(A) :- contains(A,B) & affects(B,C) & pain(C)
sedative(A) :- contains(A,B) & affects(B,C) & excitement(C)
active(A) :- affects(A,B)
add_on(A) :- contains(B,A), placebo(B) with warning that this does not cover all tuples
placebo(A) :- not(monodrug(A)) & not(combidrug(A))
combidrug(A) :- not(monodrug(A)) & anodyne(A) with warning
monodrug(A) :- not(combidrug(A)) & anodyne(A) with warning
```

The definitions of monodrug, combidrug, and placebo cannot be found by FOIL. The rules found do not cover all positive instances. It is easily seen that the cross-classification leads to some confusion: FOIL tries to use anodyne for the definition of combidrugs and monodrugs. This, however, does not lead to the formation of an MSG. The learning result of KLUSTER is a different representation formalism and requires different inputs than FOIL.[6] The main difference between KLUSTER and FOIL, however, concerns the search in the hypothesis space. Whereas KLUSTER can construct a consistent MSG if one exists, FOIL's search heuristics cannot guarantee finding a hypothesis that is consistent with the data, because an encoding of the SAT problem (Garey & Johnson, 1979) is a possible learning problem of FOIL but not of KLUSTER (cf. Haussler, 1989; Kietz, 1993). Since we know that SAT is intractable, any equivalent learning problem is intractable as well.

4.2.3. Constructive induction

Approaches to constructive induction can be structured with respect to the reasons for introducing a new term. KLUSTER's reason for introducing a new term is the need to refer to a particular set of objects. This need is constituted by the definition of another concept. KLUSTER also introduces new relations, as was shown in our example of section 3. The newly introduced terms are specializations of already given or learned terms of the hypothesis language.

The CIGOL system, which implements induction as inverse resolution, learns literals that can play the role of a missing premise, given the other premises and the conclusion from a resolution step (Muggleton & Buntine, 1988). CIGOL introduces new terms into the hypothesis language. The decision as to whether the newly introduced term should be kept or removed is left to the user. Therefore, no criterion for the selection of a new term is formalized. Moreover, the search space for a new literal is $2^n - 1$, given a substitution with n elements. In the literature on inverse resolution, there exists no formalized method to focus the search within this space. Finally, CIGOL does not define the newly introduced predicate. KLUSTER's introduction of a new concept can be viewed as learning a missing premise of a classification rule. However, the implemented method is more efficient than inverse resolution because the search space is limited and the search within it is focused. In KLUSTER, at most n concepts can be newly introduced, given n relevant roles. A new term is only introduced into the hypothesis language if KLUSTER is capable of defining it.

5. Conclusion

KLUSTER is the first learning algorithm that is capable of learning a concept structure in the framework of term subsumption formalisms. Concepts are defined by relations to other concepts that are uniformly represented within the same concept structure. Thus, a learned concept or role serves to define another concept. There is no separation between background knowledge and learned knowledge. Concepts are represented in a structure involving several roots. Cross-classification or forming subconcepts under diverse aspects is possible in KLUSTER. The interrelatedness of concepts is expressed not only by the concept representation but also by the way concepts are learned. Concepts are formed in the context of mutually disjoint concepts (MDCs). Refinements of concepts and roles are made in the course of defining a concept. In this way, the KLUSTER approach represents and exploits a rich concept structure.

KLUSTER learns most specific generalizations (MSGs) as well as most general discriminations (MGDs). With respect to a particular representation, it is guaranteed that KLUSTER will find the unique MSG in polynomial time. Finding the best MGD would be exponential, so KLUSTER takes the first MGD found. The introduction and definition of new roles potentially makes classification exponential. Therefore, defined roles are excluded from the basic algorithm. Only some defined roles are introduced if they are really needed for the distinction between concepts whose extensions are disjoint. Learning new roles is polynomially bounded by two parameters. KLUSTER inductively learns in polynomial time. The use of KLUSTER's learning results (i.e., the deductive classification) cannot be performed completely in polynomial time because of the defined roles.

Acknowledgments

The authors thank Tom Dietterich and Ross Quinlan wholeheartedly for valuable comments. Christof Peltason should also be mentioned for his concern about term subsumption languages and the BACK system. Part of this work is partially funded by the CEC, ESPRIT P2154.

Notes

1. A recent approach to learning in the term-subsumption formalism is (Cohen & Hirsh, 1992).
2. For a discussion of the computational complexity of entailment, see Nebel (1990, section 4.5).
3. It is the user who names root concepts; the system generates an artificial name such as `rootconcept_1`. Primitive concepts are named based on the names in the ABox.
4. Computing MDCs from pairwise disjointness of concepts corresponds to the NP-complete problem CLIQUE (Garey & Johnson, 1979). But for KLUSTER the inheritance in the basic taxonomy restricts the number of concepts (nodes) in one CLIQUE. In the example, at most five of all twelve concepts are to be considered as a CLIQUE: the concepts subsumed by `drug`.
5. FOIL has two modes, one with negated and one without negated literals in rule premises. We ran FOIL in both modes and show the best rules of both runs.
6. When trying out KLUSTER on the senator votes domain, KLUSTER detected that the Democratic senators all voted for South Africa sanctions, whereas there was no topic on which the Republican senators all gave the same vote.

References

Bisson, G. (1990). KBG, a knowledge-based generalizer. In *7th ICML-90* (pp. 9–15). Morgan Kaufmann.

Borgida, A., Brachman, R.J., McGuinness, D.L., & Resnick, L.A. (1989). Classic: a structural data model for objects. *Proceedings of ACM SIGMOD-89* (pp. 58–67). Portland, OR.

Brachman, R.J., and Schmolze, J.G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science, 9*, 171–216.

Brachman, R.J. (1977). What's in a concept: structural foundations for semantic networks. *International Journal of Man-Machine Studies, 9*, 127–152.

Brachman, R.J., Gilbert, V.P., & Levesque, H.J. (1985). An essential hybrid reasoning system. In *IJCAI-85* (pp. 532–538), Morgan Kaufmann.

Buntine, W. (1988). Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence, 36*, 149–176.

Ceri, S., Gottlob, G., & Tanca, L. (1990). *Logic programming and databases*. New York: Springer.

Cohen, W.W., Borgida, A., & Hirsh, H. (in press). Computing least common subsumers in description logic. *Proceedings of AAAI-92*.

Cohen, W.W., & Hirsh, H. (1992). Learnability of description logics. *Proceedings of the Fourth COLT*, ACM Press, pp. 116–127.

Donini, F.M., Lenzerini, M., Nardi, C., & Nutt, W. (1991). Tractble concept languages. *Proceedings IJCAI-91* (pp. 458–463).

Emde, W., Habel, C. & Rollinger, C.R. (1983). The discovery of the equator or concept-driven learning. *Proceedings IJCAI-83* (pp. 455–458), Morgan Kaufmann.

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

Garey, M.R., & Johnson, D.S. (1979). *Computers and intractability—A guide to the theory of NP-completeness*. New York: Freeman.

Haussler, D. (1989). Learning conjunctive concepts in structural domains. *Machine Learning, 4*, 7–40.

Kearns, M.J. (1990). *The computational complexity of machine learning*. Cambridge, MA, London: MIT Press.

Kietz, J.-U. (1992). A comparative study of structural most specific generalizations used in machine learning. *Proceedings of ECAI'92 Workshop W18*.

Kietz, J.-U. (1993). Some lower bounds for the computational complexity of inductive logic programming. *Proceedings of Machine Learning ECML-93*. Berlin: Springer, pp. 115–123.

Kietz, J.-U., & Wrobel, S. (1991). Controlling the complexity of learning in logic through syntactic and task-oriented models. *Proceedings of Inductive Logic Programming Workshop*, Porto. Also in S. Muggleton (Ed.) (1992). *Inductive logic programming*. New York: Academic Press, pp. 311–333.

Kietz, J.-U. (1988). Incremental and reversible acquisition of taxonomies. *Proceedings of the European Knowledge Acquisition Workshop*. Birlinghoven: GMD-Studien No. 143.

Kodratoff, Y., and Tecuci, G. (1989). The central role of explanations in DISCIPLE. In K. Morik (Ed.), *Knowledge representation and organization in machine learning*. New York: Springer, pp. 135–147.

Lebowitz, M. (1987). Experiments with incremental concept formation: UNIMEM. *Machine Learning, 2*, 103–138.

Luck, K.V., Nebel, B., Peltason, C., & Schmiedel, A. (1987). The anatomy of the BACK system (KIT-Report No. 41). Berlin: Technical University Berlin.

Michalski, R.S. (1983). A theory and methodology of inductive learning. In *Machine learning—An artificial intelligence approach* (Vol. I). Los Altos, CA: Morgan Kaufmann, pp. 83–134.

Michalski, R.S. (1990). Learning flexible concepts: fundamental ideas and a method based on two-tiered representation. In Y. Kodratoff & R.S. Michalski (Eds.), *Machine Learning—An artificial intelligence approach* (Vol. III). San Mateo, CA: Morgan Kaufmann, pp. 63–111.

Morik, K., & Kietz, J.-U. (1989). A bootstrapping approach to conceptual clustering. *Proceedings of the Sixth International Workshop on Machine Learning*, Morgan Kaufmann.

Morik, K., Wrobel, S., Kietz, J.-U., & Emde, W. (1993). *Knowledge Acquisition and Machine Learning—Theory, Methods, and Applications*. London: Academic Press.

Moser, M.G. (1983). An overview of NIKL, the new implementation of KL-one. In *Research in knowledge representatin and natural language understanding*. Cambridge, MA: B. Beranek and Newman Inc.

Muggleton, S., and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. *Proceedings of IWML-88*. Ann Arbor, MI: Morgan Kaufmann.

Muggleton, S. (1990). Inductive logic programming. *Proceedings of the First Conference on Algorithmic Learning Theory*. Tokyo: Ohmsha.

Muggleton, S. & Feng, C. (1990). Efficient induction of logic programs. *Proceedings of the First Conference on Algorithmic Learning Theory*. Tokyo: Ohmsha.

Nebel, B. (1990). *Reasoning and revision in hybrid representation systems*. New York: Springer.

Peltason, C., Luck, K., & Kindermann, C.K. (1991). Terminological logic users workshop (KIT-Report 95). Berlin: Technical University Berlin.

Peltason, C., Schmiedel, A., Kindermann, C., & Quantz, J. (1989). The BACK System revisited (KIT-Report 75). Berlin: Technical University Berlin.

Plotkin, G.D. (1970). A note on inductive generalization. *Machine Intelligence, 5*, 153–163.

Quinlan, R. (1990). Learning logical defnitions from relations. *Machine Learning, 5*, 239–266.

Shapiro, E. (1983). *Algorithmic program debugging*. Cambridge, MA: MIT Press.

Stepp, R.E. & Michalski, R.S. (1986). Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R. Michalski, J. Carbonell, & T. Michell (Eds.), *Machine learning—An AI approach* (Vol. II). San Mateo: Morgan Kaufmann, pp. 471–498.

Thompson, K., & Langley, P. (1989). Incremental concept formation with composite objects. *Proceedings of the 6th International Workshop on Machine Learning*, Morgan Kaufmann, pp. 373–374.

Vilain, M. (1985). The restricted language architecture of a hybrid reasoning system. *IJCAI-85* (pp. 547–551).

Wrobel, S. (1987). Higher-order concepts in a tractable knowledge representation. In K. Morik (Ed.), *Proceedings of the German Workshop on Artificial Intelligence*. Berlin: Springer, pp. 129–138.

Received January 16, 1992
Accepted July 24, 1992
Final Manuscript September 17, 1992