MICSL: Multiple Iterative Constraint Satisfaction based Learning

George Potamias [1,2]

[1]Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH),
P.O. Box 1385, GR- 711 10 Heraklion, Crete, Greece

[2]Department of Computer Science, University of Crete,
P.O. Box 1470, GR- 714 09, Heraklion, Crete, Greece

E-mail: potamias@ics.forth.gr

Abstract

A novel concept learning algorithm named, MICSL: Multiple Iterative Constraint Satisfaction based Learning, is presented. The algorithm utilizes mathematical programming and constraint satisfaction techniques towards uniform representation and management of both data and background knowledge. It offers a flexible enough learning framework and respective services. The representation flexibility of MICSL rests on a method that transforms propositional cases, represented as propositional clauses, into constraint equivalents. The theoretical background as well as the validity of the transformation process are analyzed and studied. Following a 'general-to-specific' generalization strategy the algorithm iterates on multiple calls of a constraint satisfaction process. The outcome is a consistent set of rules. Each rule composes a minimal model of the given set of cases. Theoretical results relating the solutions of a constraint satisfaction process and the minimal models of a set of cases are stated and proved. The performance of the algorithm on some real-world benchmark domains is assessed and compared with widely used machine learning systems, such as C4.5 and CN2. Issues related to algorithm's complexity are also raised and discussed.

Keywords: Machine learning, concept learning, mathematical programming, constraint satisfaction.

1. Introduction

Concept learning from examples (CLFE) is one of the most experienced paradigms of machine learning research. A variety of different CLFE techniques have appeared in the literature. The list includes decision tree induction, instance-based learning, connectionist architectures, model-based learning, and others. Most of the approaches tackle the problem of *supervised* CLFE where, concept descriptions are formed by observing and analyzing a set of examples each of which is pre-assigned to a *category* (or, *class*). The sets of classes are the concepts to be defined. Usually, the definition of a class is described in terms of a set of attribute-value pairs.

The *attribute-value* representation scheme shares a number of advantages but it exhibits a number of disadvantages as well. On one hand, attribute-value representation renders to the device of algorithmic learning processes that exhibit relatively low computational-time performance. Respective algorithms like ID3 [45] - a decision tree induction system, CN2 [16] - a straightforward rule-learning system or, IBL [1] - a family of instance based learners, have proved their efficient concept learning ability. On the other hand, attribute-value is by its nature a *flat* representation model. The main disadvantage of flat models is the restriction they pose on the ability to represent complex and rich data and knowledge structures. Indeed, the utility of *background*, or *prior* knowledge incorporation in learning processes has been recognized in many theoretical and applied studies [17, 23, 49]. In many cases useful information is available, and even not necessarily correct, it can beneficially be used as a *learning bias* to supplement inductive mechanisms. This is the situation in many real life applications as for example in medical decision making where, domain theories could enhance, and even reform learning results, making them more comprehensible to the field experts and practitioners. Of course, there is always the alternative to follow a first-order logic representation scheme and use respective relational learning, e.g., Mobal [36] or, inductive logic programming systems [37, 38]. However, such systems are pre-determined to face and deal with heavy computational-time requirements.

In the present study a novel concept learning algorithm namely, *MICSL: Multiple Iterative Constraint Satisfaction based Learning*, is presented. The algorithm is based on the representation scheme offered by *mathematical programming* formalism and techniques. In order to overcome the limitations imposed by attribute-value representation and provide a general-purpose machine learning environment, an *iterative constraint satisfaction* learning framework is elaborated. The algorithm is a straightforward utilization of *linear programming* techniques appropriately adjusted to the CLEF framework. In this context, optimization is viewed as a natural companion to induction and machine learning. With this respect, induction of a rule that fits a given set of training cases resembles the process of finding and forming a function that obeys certain criteria [2, pp. 156]. The function to form corresponds to the disjunction of all *optimal* attribute-value combinations that comply with the given constraints. The criteria to obey correspond to a set of *constraint equivalents* of the given propositional cases.

Interest on utilizing mathematical programming, i.e., optimization subject to constraints, techniques in the context of machine learning and data mining has recently increased [30, 31, 32]. Mathematical programming (MP) has a long history in decision and computational sciences. In its broad discipline MP serves a number of research areas and exhibits both theoretical and applied achievements [53]. The representation and formulation of mathematical programming endeavors split into two main categories, *linear-programming*, and non-linear programming. Here we concerned with problems sharing a linear-, actually integer-programming formulation which, in general, are more easily tackled and solved [40]. The general setting of a *linear-programming problem* (LPP) is shown in Figure 1.

LPP ***minimize*** (or, ***maximize***) $\boldsymbol{f}(x_1,x_2, \dots , x_n)$

subject ***t***o ***c***onstraints (***s.t.c.***)

$$\boldsymbol{g_c}(x_1,x_2, \dots , x_n) \leq (\geq) \ \boldsymbol{b}$$
$$x_1,x_2, \dots , x_n \in \boldsymbol{D}$$

- $\boldsymbol{f}$, is a *linear function*. A function $\boldsymbol{f}(x_1,x_2, \dots , x_n)$ is a linear function if and only if for some set of constants, $c_1,c_2, ..., c_n$, $\boldsymbol{f}(x_1,x_2, \dots , x_n) = c_1 x_1 + c_2 x_2 + ... + c_n x_n$
- $\boldsymbol{g_c}$ on any number $\boldsymbol{b}$, is a *linear inequality*, and $\boldsymbol{c}$ equals the number of constraints
- $\boldsymbol{D}$, is the *domain* of variables $x_1, x_2, \dots, x_n$. If D is the set of integers, then we have an *integer-programming problem* (IPP), and if D is restricted to set $\{0,1\}$, then we have a *{0,1}-programming problem*. (or, *binary-programming problem*)

Figure 1. Setting of a linear-programming problem

The use of linear programming techniques in the context of machine learning focuses mainly on discrimination analysis (DA) tasks. The respective approaches study the difference between two or more sets of *m*-dimensional points based on one or more attributes. The solved LPP is then used to classify new observations into a group to which they are likely to belong [21].

Solving a LPP is equivalent with the goal of assigning values from domain *D* to its variables, i.e., *labeling* of the variables, in a way that, (1) the objective function is maximized, or minimized, and (2) all of the given constraints are *satisfied*. If the objective function *f* is missing then, the goal is to satisfy just the given set of constraints. In this setting, a *constraint satisfaction problem* (CSP) has to be solved [51].

Successful constraint satisfaction techniques are mostly utilized in the *constraint logic programming* (CLP) context. CLP renders efficient and powerful formalisms and respective logic programming frameworks [23, 26]. The MICSL algorithm utilizes and exploits constraint satisfaction operations in a CLP framework. The extended constraint logic programming language *Eclipse* [19, 20], was used as the implementation platform of the algorithm.

The main contribution of the MICSL algorithm comes from its constraint-based representation scheme. Following a well-defined procedure propositional cases are transformed into *constraint equivalent* forms. Domain specific or, other exploratory data analysis requirements could be also transformed to equivalent constraints. The search space heuristic of the algorithm is realized by *iterative* calls to a respective *constraint satisfaction* (CS) operation. The CS operation functions on the set of knowledge and data constraint equivalents. Viewing inductive concept learning as a search through a large space of hypotheses, the goal is to find hypotheses that *optimally* fit the training set of cases. A hypothesis is interpreted as optimal if it presents a *minimal* model of the given propositional cases. The underlying operations are based on appropriately formed optimization requests to be performed by the CS process. The optimization requests follow a *general-to-specific* ordering, as stated in [35, pp. 24-26], of the potential fitting hypotheses. So, the implemented *generalization* strategy searches for less, to more complex rules.

Next section presents a technique that transforms propositional cases into linear constraints. In section 3 the MICSL algorithm is presented and its theoretical basis is formally stated and justified. Section 4 reports on experimental results; comparative results with reference machine learning algorithms are also presented. A comparative presentation of the related work is given in section 5. The last section is devoted to concluding remarks and hints for future research.

2. From Propositional Examples to Linear Constraints

In this section the fundamentals of transforming a set of propositional cases into a corresponding set of linear constraints are presented. Then, optimization techniques are appropriately reformulated and adjusted to serve *generalization* and *inductive* inference. Previously reported work, [3], simulates *deductive* inference using similar optimization techniques. The presentation follows the Propositional Logic representation formalism.

Let $\mathcal{L}$ be a language that contains finitely many constants, used to denote the attribute- and class-values of a training set of cases E (the language is supposed not to contain any functional symbol). With $E_{\mathcal{L}}$ we denote the set of all attribute- and class-values of E. Assume a domain with m attributes and k classes.

Definition 1. A *propositional case*, e, is of the form:

$$e \equiv C_c \leftarrow A_{1;v(1)} \wedge A_{2;v(2)} \wedge \ldots \wedge A_{m;v(m)}$$

where, $C_c, A_{i;v(i)} \in E_{\mathcal{L}}$ are all literals; C_c, $1 \leq c \leq k$, represents a class-value; $A_{i;v(i)}$ represents the value of attribute A_i in case e, $1 \leq v(i) \leq \#Att\_A_i\_values$, $1 \leq i \leq m$.

Definition 2. Let X be a variable. X is called a *binary variable* if it takes on only the values *0* and *1*. Furthermore, assume a domain with m attributes and k class-values where, attribute A_i, $1 \leq i \leq m$, takes values from set $\{A_{i;1}, A_{i;2}, \ldots A_{i;v(i)}\}$, and class C from $\{C_1, C_2, \ldots C_k\}$; $A_{i;j}, C_c \in E_{\mathcal{L}}$, $1 \leq i \leq m$, $1 \leq j \leq v(i)$, $1 \leq c \leq k$. We let $X_c, X_{i;v(i)}$ be the binary variables corresponding to class- and attribute-values, C_c and $A_{i;v(i)}$, respectively. The set $\{X_c, X_{i;v(i)} \mid C_c, A_{i;v(i)} \in E_{\mathcal{L}}\}$ will be called the *binary variable representation* of $E_{\mathcal{L}}$.

The binary variable $X_{i;j}$, represents the truth-value of $A_{i;j}$, i.e., $A_{i;j}$ holds. Thus, the truth value of $\neg A_{i;j}$, the negation of $A_{i;j}$, is represented by, $(1\text{-}X_{i;j})$, i.e., $A_{i;j}$ does *not* hold.

Example (Examples to constraints]). Consider the following set of training cases, which defines the logical *AND* operator.

$$C_1 \leftarrow A_{1;1} \wedge A_{2;1}$$
$$C_2 \leftarrow A_{1;1} \wedge A_{2;2}$$
$$C_2 \leftarrow A_{1;2} \wedge A_{2;1}$$
$$C_2 \leftarrow A_{1;2} \wedge A_{2;2}$$

Take the first case, C is `true` ($C = C_1$) if A_1 and A_2 are both `true` ($A_1 = A_{1;1}$ and $A_2 = A_{2;1}$). With the appropriate binary variable representatives for $Ai;j$, the following constraint equivalents could be formed,

$$X_1 \geq X_{1;1} * X_{2;1}$$
$$X_2 \geq X_{1;1} * X_{2;2}$$
$$X_2 \geq X_{1;2} * X_{2;1}$$
$$X_2 \geq X_{1;2} * X_{2;2}$$

Take the first constraint; if $X_{1;1} = X_{2;1} = 1$ then, $X_1 \geq 1$. But X_1 is a binary variable, so it is set equal to *1*, i.e., the corresponding class-value C_1 is `true`. In other words, the aforementioned constraints are well defined, and soundly encode the underlying *truth* of the given cases.

The following definition generalizes on the previous example and provides the formalism for transforming propositional cases into constraints.

Definition 3. Let E be a set of propositional cases $e \in E$, $e \equiv C_c \leftarrow A_{1;v(1)} \wedge A_{2;v(2)} \wedge \ldots \wedge A_{m;v(m)}$, for a domain described by m attributes. The *constraint version* of e, denoted by $c(e)$, is presented by inequality:

$$X_c \geq \prod_{i=1}^{m} X_{i;v(i)} \quad (1)$$

where, X_c, $X_{i;v(i)}$ are the binary representatives of C_c and $A_{i;v(i)}$, respectively. The *constraint version* of E, denoted by $c(E)$, is the set of constraint versions of all propositional cases in E.

The constraint versions of the propositional cases in Definition 3 (inequality 1) are nonlinear, which are generally more difficult to solve than linear constraints. The following lemma remedies this and *linearizes* the constraints in $c(E)$. The lemma is based on a notion of equivalence between sets of constraints, with respect to the domain of the involved variables.

Definition 4. Let D be a domain of variables. Two sets $c(E_a)$ and $c(E_b)$ of constraints are *D-equivalent* if and only if every solution that assigns values from D to the variables of $c(E_a)$ is a solution of $c(E_b)$, and vice versa.

Lemma 1. (Bell et. al., 1996, [3]) Let X, X_1, ... X_m be binary variables. The constraint $X_c \geq \prod_{i=1}^{m} X_i$ is *{0,1}-equivalent* to the constraint $X_c \geq 1 - \sum_{i=1}^{m} (1 - X_i)$

Based on the above lemma, and after some calculations, nonlinear constraint (1) transforms into the linear constraint,

$$\sum_{i=1}^{m} X_{i;v(i)} - X_c \leq m - 1 \quad (2)$$

2.1. Constraint-based representation and complex data structures

One of the main challenges in machine learning research is how to represent, and deal, with *structured* concepts, i.e., concepts following a *hierarchical* definition or, *relational* concepts. Moreover, it is well known that *background*, or *prior knowledge* can improve learning. The basic idea is to realize the *declarative bias*, introduced by background knowledge, in a way that, (1) the hypotheses space is restricted, resulting into more efficient and effective learning operations [17, 50], (2) intermediate concepts, that lay between observables and target concepts, are utilized and even inductively constructed (for example, the work in theory revision [41]), and (3) the general form of induced hypotheses match a pre-specified template [36, 49].

The constraint-based scheme is flexible enough to allow complex structure representation.

- *Relations*: Take as example a particular application domain where the concept `polygon` is defined in terms of the two sub-concepts, `square` and `triangle`. A rule construct could be formed: `P(olygon)` $\leftarrow$ `S(quare)` $\vee$ `T(riagle)`, stating the relation holding between the concepts. Setting the binary variable representatives X_P, X_S and X_T for `P(olygon)`, `S(quare)` and `T(riangle)`, respectively, and based on the aforementioned general constraint form (2), the constraint, $X_S + X_T - 2\,X_P \leq 0$ could be formed to encode the stated relation.

- *Mutual-exclusiveness*: An implicit assumption made when applying various machine learning algorithms on a set of propositional cases, is the *mutual-exclusiveness* between the values of the same attribute. Assume that attribute A_i takes on values from set $\{A_{i;1}$,

$A_{i;2}$, …, $A_{i;v(i)}\}$ then, the constraint, $\sum_{i=1}^{v(i)} X_{i;v(i)} \leq 1$ encodes the mutual exclusiveness requirement.

- *Negation*: As it is noted above, $1\text{-}X_{i;j}$ represents the fact that the respective attribute-value proposition $A_{i;j}$ is `false`. So, negative information, present in the facts of a particular application domain, could be uniformly and easily coded and represented.

- *Unknown i*nformation: A valuable characteristic of a learning algorithm is its ability to represent and treat the unknown status of attributes. Unknown information could be represented as a *disjunctive* fact namely, the disjunction of attribute's values. Consider a hypothetical propositional case (for a domain with three attributes), $C_c \leftarrow A_{1;3} \wedge Unknown \wedge A_{3;2}$ where, the value of attribute A_2 is unknown. Assume that attribute A_2 takes on values from set $\{A_{2;1}, A_{2;2}, \ldots, A_{2;v(2)}\}$, with $v(2)$ the total number of values for attribute A_2. An additional *dummy* propositional attribute-value, $A_{2;u}$, is introduced. It represents the unknown information status of this attribute. The constraint presented by inequality, $X_{2;1} + X_{2;2} + \ldots + X_{2;v(2)} - v(2)*X_{2;u} \leq 0$, defines the unknown attribute-value with reference to attribute's known values. If mutual-exclusiveness between attribute-values is assumed then, $v(2)$ should be set equal to 1, otherwise, it should be set equal to the number of allowable values. With the appropriate definition of the unknown dummy variable, the constraint version of the given hypothetical case takes the form, $X_{1;3} + X_{2;u} + X_{3;2} - X_c \leq 2$.

In general, a rich family of propositional complex constructs could be transformed and expressed into their suitable and valid constraint versions. As a reference to some basic and interesting transformations, the validity of the following transformations, reported in [54], could be easily checked (X, X_i are binary variables).

$$\bigwedge_{i=1}^{m} X_i \rightarrow X \;\Rightarrow\; \sum_{i=1}^{m} X_i - X \leq m - 1$$

$$\bigvee_{i=1}^{m} X_i \rightarrow X \;\Rightarrow\; \sum_{i=1}^{m} Xi - mX \leq 0$$

$$\bigwedge_{i=1}^{m} X_i \;\; is \;\; true \;\Rightarrow\; \sum_{i=1}^{m} X_i \geq m$$

$$\bigwedge_{i=1}^{m} X_i \;\; is \; false \;\Rightarrow\; \sum_{i=1}^{m} X_i \leq m - 1$$

$$\bigvee_{i=1}^{m} X_i \;\; is \; true \;\Rightarrow\; \sum_{i=1}^{m} X_i \geq 1$$

$$\bigvee_{i=1}^{m} X_i \;\; is \; false \;\Rightarrow\; \sum_{i=1}^{m} X_i \leq 0$$

The constraint-based representation scheme is flexible enough to represent and treat structured or relational information presented as: rule schemata and templates, integrity constraints, or other, more complex forms of background knowledge and declarative bias requirements. The question that now raises is how the constraint-based representation is utilized to simulate generalization processes and inductive inference.

3. Constraint Satisfaction based Learning

The operations implemented by the MICSL algorithm are presented below via the simple four cases Example presented in the previous section. The algorithm implements six- (6) steps and its final output is a set of *rules* that optimally fit the data. In steps (1) and (2), the transformation of cases into constraints is performed and a respective constraint satisfaction problem is formed. Steps (3) to (6), implements the inductive kernel operations of the algorithm.

Step 1. *THEN part & mutual-exclusive class-values*: A rule should conclude into a class-value. So, the binary variable representative of one of the class-values should be equal to *1*. Furthermore, class-values are considered to be mutual-exclusive. The constraint realizing these requirements takes the form,

c(1): $X_1 + X_2 = 1$

For a domain with *k* class-values the respective constraint form is,

c(Class): $\sum_{i=1}^{k} X_i = 1$

Step 2. *Mutual exclusive attribute-values*: In most machine learning applications the values of an attribute are considered to be mutual-exclusive. The constraints corresponding to this requirement are,

c(2): $X_{1;1} + X_{1;2} \leq 1$

c(3): $X_{2;1} + X_{2;2} \leq 1$

In a domain described by *m* attributes, and *v(i)* values for each attribute A_i, the respective constraints takes the following general form,

c(A_i): $\sum_{j=1}^{v(i)} X_{i;j} \leq 1$

Step 3. *IF part - Setting the generalization parameter*: MICSL searches the hypotheses' space for less, to more complex rules. So, a parameter *R* should be set in order to control the required number of attribute-values in the *if* part of a rule. The corresponding constraint is,

c(4): $X_{1;1} + X_{1;2} + X_{2;1} + X_{2;2} = R$

In the presence of *m* attributes the general form of the constraint is,

c(OF): $\sum_{i=1}^{m} \sum_{j=1}^{v(i)} X_{i;j} = R$

That is, at most *R* binary variables should be equal to *1*, and by though, at most *R* attribute-values should be present in the *if* part of a rule. This constraint compose the *objective function* of the *linear*, actually *{0,1}-programming problem* to be formed. Depending on syntactic and semantic specifics of the application domain or other, user specified data exploration requirements, the value of *R* may be initialized to a bigger value (e.g., the user may be interested to induce rules with more, than just 1, attributes in their if part). This unique characteristic offered by the constraint-based representation framework, makes machine learning operations flexible enough to adapt to complex exploratory data analysis tasks.

Step 4. *Generalization step & Parameterization of constraint versions of cases*: The satisfiability of a case is subject to the number of attributes, and to the specific attribute-values used to describe it. For example, $A_{1;1}$, (as well as $\{A_{2;1}\}$, and $\{A_{1;1}, A_{2;1}\}$), satisfies the first case, as it is validated by the following observation: with the constraint version of the cases, and applying a constraint satisfaction operation, the solution, $X_{1;1} = X_1 = 1$ could be found.

This solution corresponds to rule, "$A_{1;1} \rightarrow C_1$" which, compose an *over-generalization* and actually missclasifies case 2 as C_1. A way to overcome this problem is to form the constraint version of cases with respect to the generalization parameter R. So, at the generalization step $R=1$, the respective constraint version of the cases forms as follows:

$\boldsymbol{c(R_{=1},5)}$: $X_{1;1} + X_{2;1} - X_2 \leq R\text{-}1 = 0$
$\boldsymbol{c(R_{=1},6)}$: $X_{1;1} + X_{2;2} - X_1 \leq R\text{-}1 = 0$
$\boldsymbol{c(R_{=1},7)}$: $X_{1;2} + X_{2;1} - X_1 \leq R\text{-}1 = 0$
$\boldsymbol{c(R_{=1},8)}$: $X_{1;2} + X_{2;2} - X_2 \leq R\text{-}1 = 0$

In the general, case and following formula (2) presented in the previous section, the constraint version of cases takes the following general form:

$\boldsymbol{c(Cases)}$: $\sum_{i=1}^{m} X_{i;v(i)} - X_c \leq R - 1$

The constraints *c(1)-c(4)* and $c(R_{=1},5)$ - $c(R_{=1},8)$, compose the {0-1}-programming problem to be solved; refer to it as *CSP(R,S)* (CSP(1,1) if R=1) where, *S* holds and denotes the set of solutions found so far.

It is unclear how to set up an objective function so that all minimal models of a set of clauses can be computed in a 'single-step' optimization. For this, an *iterative*, 'multi-step' algorithm is used, [3], that computes all minimal models of a disjunctive deductive database. Recall that by Definition 1, the form of a propositional case is indeed a clause form.

Algorithm (Minimal Models).

Denote with *CSP(R,S)* the current constraint satisfaction problem to be solved; at iteration *R* and *S* solutions found so far.

i) Set $S = \varnothing$
ii) Solve *CSP(R,S)* i.e., by calling a constraint satisfaction routine on the formed set of constraints
i) If no (optimal) solution can be found, halt and return *S* as the set of minimal models
ii) Otherwise, let M_S be the minimal model corresponding to the solution found in Step ii. Add M_S to *S*.
iii) Form and add the constraint $\boldsymbol{c(M_S)}$: $\sum_{X_S \in M_S} X_S \leq |M_S| - 1$ where, $|M_S|$ is the number of positive valued ($=1$) binary variables of the current solution. Set $S' = S \cup M_S$, and $CSP(R,S') = CSP(R,S) \cup c(M_S)$. Go to step ii.

The above algorithm could be easily customized to serve an inductive generalization operation.

Step 5. *Finding solutions & Forming rules*: Following step ii of the above algorithm, i.e., calling a constraint satisfaction routine on $CSP(1, \varnothing)$, the first solution found is: $X_{2;2} = X_2 = 1$. Each potential solution is a vector of *1*s and *0*s, corresponding to the `true` and `false` states of specific attribute- and class-values, respectively. Furthermore, just $R+1$ number of *1*s will occur in each of the solutions, as required by *c(OF)* and *c(Class)* constraints (formed in steps 3 and 4). Keeping just the *1*s, $M_1 = \{X_{2;2}\}$ compose the minimal model corresponding to the found solution. The *conjunction* of the corresponding attribute-values may be interpreted as an inferential *rule* concluding to the respective class-value. The above solution translates into the following rule,

$$\boldsymbol{r_1}: A_{2;2} \rightarrow C_2$$

which is a valid inferential rule stating: "*if* A_2 *is* `false` *then,* $C = (A_1$ *AND* $A_2)$ *is also* `false`". Set $S_1 = \{r_1\}$.

Step 5.1. *Adding constraints for found solutions*: Following again step ii, the following constraint is formed and added,

$$\boldsymbol{c(r_1)}: X_{2;2} \leq 0$$

With this constraint added, the solution found will not be part of any other solution that potentially will be found in subsequent steps. So, *minimality* of solutions is guaranteed (see section 3.1 for a justification of this statement).

Step 5.2. *More solutions*: Remaining within step ii, set $CSP(1,S_1) = CSP(1, \varnothing) \cup \{c(r_1)\}$. The constraint satisfaction routine is called again on the newly formed constraint satisfaction problem. The solution, $X_{1;2} = X_2 = 1$ is found and the following rule is formed,

$$\boldsymbol{r_2}: A_{1;2} \rightarrow C2$$

which is also a valid, "*if A_1 is* `false` *then, $C = (A_1$ AND $A_2)$ is also* `false`". The following constraint is formed and added,

$$\boldsymbol{c(r_2)}: X_{1;2} \leq 0$$

Set $S_2 = \{r_1, r_2\}$.

Step 5.3. *Stopping condition*: Trying to solve $CSP(1,S_2) = CSP(1,S_1) \cup \{\boldsymbol{c(r2)}\}$ no solution is found. If $R < m$, where m is the total number of attributes, go to step 6, otherwise ***stop***.

Step 6. *Iterate*: Set $R = R+1$ and go to step 3. Form a new objective function and go to step 4. The set of found solutions so far is S_2. The constraint version of cases are parameterized according to the new value of R and the new constraint version of cases are formed, $CSP(2,S_2) = \{c(1), c(2), c(3), c(4), c(R_{=1},5), c(R_{=1},6), c(R_{=1},7), c(R_{=1},8)\}$. Go to step 5. The solution found is, $X_{1;1} = X_{2;1} = X_1 = 1$, which translates into the following rule:

$$\boldsymbol{r_3}: A_{1;1} \wedge A_{2;1} \rightarrow C_1$$

i.e., "*if A_1 and A_2 are both* `true` *then, $C = (A_1$ AND $A_2)$ is also* `true`", which is also valid.

Forming and adding the respective solution a new CSP is formed. Trying to solve it, and *label* the involved binary variables, no solution is found. The process terminates with three solutions found and three corresponding rules formed. It can easily be checked that the three rules are not only valid but they also compose the minimal set of rules that correctly define the logical *AND* connective.

In Figure 2, a pseudo-code of the basic MICSL operations is shown.

```
Set m to be the number of attributes, and k the number of class values
    Form_Atts_Class_Constraints(m,k)
    Set R = 1 , S= ∅
        while R ≤ m
            do
            Find_All_Solutions(R,S)
            Set R = R+1

    Find_All_Solutions(R,S)
        Form_CSP(R,S)
            repeat
            solve(CSP(R,S))
                if solution_found(s)
                Update_Set_of_Found_Solutions(S'), S' = S U {s}
                else
            exit
```

Figure 2. The MICSL algorithm

3.1. Theoretical basis of MICSL

The power of the MICSL algorithm is based on its ability to identify all the *minimal models* of a set of propositional cases. This fact is supported and validated by theoretical results that relate the solutions to the constraint equivalents of cases, *c(E)*, with the models of *E*.

A model of *E* is a set of *positive* occurrences of the given attribute- and class-values. We are interested for those models of *E* which, (1) assign the `true` value to an *optimal* number of attribute-values, and (2) always assign the `true` value to just one of the given class-values. Most of the results that follows correspond to related results reported in [3]. The difference is that, the respective constraint-satisfaction deductive inferential framework presented in [3], is appropriately adjusted to serve generalization operations and *inductive inference*.

Definition 5. Let *I(E)* be a model of a set of propositional cases *E*. Define the *binary variable assignment* $S_{I(E)}$ corresponding to *I(E)* as follows:

$$\forall A_{i;j},\ C_c,\ S_{I(E)}(A_{i;j}\ ;\ C_c) = \begin{cases} 1 & \text{if } A_{i;\,j}\ ; C_c \in I(E) \\ 0 & \text{otherwise} \end{cases}$$

The following theorem proves a 1-1 correspondence between solutions of *c(E)*, i.e., the constraint version of *E*, and the models of *E*.

Theorem 1. *Let E be a set of propositional cases, I(E) be a model of E, and* $S_{I(E)}$ *be the binary variable assignment corresponding to I(E).* $S_{I(E)}$ *is a solution of c(E) if and only if I(E) is a model of E.*

Proof

1. ($\Leftarrow$) $S_{I(E)}$ *is a solution of c(E) if I(E) is a model of E .*

Suppose $S_{I(E)}$ is not a solution of *c(E)*. Then, a case, $e \equiv C_c \leftarrow A_{1;v(1)} \wedge A_{2;v(2)} \wedge \ldots \wedge A_{m;v(m)}$ exists for which, its constraint version $c(e) \equiv X_c \geq \prod_{i=1}^{m} X_{i;v(i)}$, is not satisfied by $S_{I(E)}$. For this to happen, it must be the case that: $S_{I(E)}(C_c) = 0$ and $\prod_{i=1}^{m} X_{i;v(i)} = 1$. So, for all $1 \leq i \leq m$, $S_{I(E)}(A_{i;v(i)}) = 1$. Then, it is true that $C_c \notin I(E)$ (by Definition 5) but, for all i, $A_{i;v(i)} \in I(E)$. Therefore *I(E)* cannot satisfy *e* and thus cannot be a model of *E*. A contradiction.

2. ($\Rightarrow$) *if* $S_{I(E)}$ *is a solution of c(E) then, I(E) is a model of E*

Suppose *I(E)* is not a model of *E*. Then a case $e \in E$ exists, which is not satisfied by *I(E)*. So, $S_{I(E)}$ cannot satisfy the constraint version of *e*. Therefore, $S_{I(E)}$ is not a solution of *c(E)*. A contradiction.

Parts 1 and 2 completes the proof of the theorem

Theorem 1, establishes the equivalence between solutions of the set of constraints *c(E)* and the models of *E*.

Now, we are in the position to state and prove the basic theorems about MICSL. First, we need a definition for minimal models and solutions. Denote, (1) by *c(R,E)*, the constraint version of *E* at iteration *R*, (2) by *I(R,E)*, the model corresponding to *c(R,E)*, and (3) by $S_{I(R,E)}$, the solution corresponding to *I(R,E)*.

Definition 6. (Minimal Models) Let *E* a set of propositional cases. *I(R,E)* is *an R-minimal model* of *E* if there exists no model *I'(R,E)* of *E* such that $I'(R,E) \subseteq I(R,E)$. *I(E)* is a *minimal model* of *E* if there exists no model *I'(E)* of *E* such that $I'(E) \subseteq I(E)$.

Lemma 2. *Let E be a set of propositional cases. At each iteration R, MICSL computes all R-minimal models of E.*

Proof

Suppose that, $S_{I(R,E)}$ is a solution of *c(R,E)*, and *I(R,E)* is the corresponding model of *E*, induced at iteration *R* (because of the equivalence between models and solutions stated by Theorem 1). Furthermore, suppose *I(R,E)* is not *R*-minimal. Then, by Definition 6, another model *I'(R,E)*, exists for which, $I'(R,E) \subset I(R,E)$. So, $\sum_{X_s \in S_{I'(R,E)}} X_s < \sum_{X_s \in S_{I(R,E)}} X_s$ holds, for the positive (equal to 1) binary variable representatives of the corresponding `true` attribute-values. But this is not possible because at each iteration *R*, only solutions with their corresponding vector values summing exactly to *R* are permitted (constraint *c(OF)*, at step 3). A contradiction. Furthermore, the algorithm iterates after all solutions summing to *R* are found (step 5). So, at each iteration *R*, MICSL computes all *R*-minimal models □

Theorem 2. *Let E be a set of propositional cases. MICSL computes all minimal models of E.*

Proof

Suppose a model *I(E)* which is not minimal. This model is induced at some iteration *R*, i.e., $I(R,E) = I(E)$. Then, by Definition 6, a model *I'(E)*, exists for which, $I'(E) \subset I(E) = I(R,E)$. Therefore, by Lemma 2, *I'(E)* should not be *R*-minimal but, a *r*-minimal model, induced at some iteration $r < R$. By step 5.1 a corresponding solution constraint is added, making $S_{I(R,E)}$ not identifiable at iteration *R*. So, $S_{I(R,E)}$ is not a solution, and $I(R,E) = I(E)$ should not be a model of *E*. A contradiction □

3.2. Constraint satisfaction based learning: global vs. greedy optimization

In this section a detailed analysis of the constraint satisfaction based learning process, as implemented by the MICSL algorithm, is presented. The analysis follows two directions: (1) MICSL is compared and contrasted with the inductive decision-tree construction approach, one of the most successful and widely used machine learning endeavors and (2) the 'robust' nature of MICSL is examined and suggestions for relaxing its 'hard' optimization heuristic are discussed.

- *MICSL & Decision-tree construction*: If one were to compare MICSL with inductive decision-tree (DT) algorithms then, DT construction should be viewed as an optimization process. DT construction processes aims to maximize the number of covered positive cases and minimize the number of negative ones. Following a *greedy* (*hill-climbing*) generalization strategy, DT construction implements a '*soft* ' optimization heuristic. Of course, the adopted greedy, hill-climbing generalization (even computationally efficient) may potentially fall in '*local minima*', ending-up to the well-known data *over-fitting* situation. Indeed, this problem composes the major intrinsic drawback of decision-tree learning processes. Various pre- or, post-pruning operations have been proposed to remedy this problem [45].

In contrast, MICSL does not implement a greedy generalization heuristic as, for example, *C4.5* does but instead, it follows a '*global*' optimization and generalization strategy. Implementing a '*hard* ' optimization heuristic, each call to the constraint satisfaction operation aims to satisfy all the given constraints and cover only positive cases. In other words the induced models are consistent with all the given cases, and each induced rule is a 100% correct. This, even if it makes MICSL a computational costly algorithm (see below for a discussion about the time-complexity issues related with the MICSL algorithm), avoids 'local minima' and escapes the data over-fitting problem. As the MICSL algorithm implements a 'general-to-specific' generalization strategy for less, to more complex rules, (i.e., by finding at each iteration *R* only the underlying *R*-minimal models) the minimality of induced rules is guaranteed.

- *Relaxing the hard optimization heuristic*: In its current setting the MICSL algorithm does not take in consideration the *importance* of attributes and their values in the course of finding solutions and forming rules, neither it computes the *significance* of rules. Computation of such measures could be utilized in order to induce and form less '*robust*' and more *informative* rules, making the final output more economic. As an example, for the Monks-1 domain, implementing a simple *filter* that excludes rules which, does not cover at least 10% of positive cases, MICSL came-up with four (4) rules (compared with the mean of 22 rules induced with-out the filter, see Table 2 in the next section). These rules capture exactly the underlying laws governing the Monks-1 domain. Here, it has to be noted that such operations act as *external post-filtering* operations that does not interfere the internal semantics of the constraint-satisfaction based learning processes.

  Of course one could interfere and change the semantics of the constraint-satisfaction process. Imposing some preference heuristics over the solutions (e.g., ordering of the solutions) could do this. In addition, a partial constraint-satisfaction structure and operation could be also accommodated. The work on *partial constraint satisfaction*, reported in [22], and on *semiring-based constraint-satisfaction*, reported in [48], compose promising alternatives towards this direction. But these methods are still pre-mature with no stable systems implementation. In the last section some hints for future improvements of the MICSL algorithm towards this direction are posted and analyzed.

3.3. Time-Complexity: a first approximation

A potential and maybe controversial drawback of the algorithm concerns its computational-time requirements. As it was already mentioned (section 1), MICSL exploits constraint satisfaction operations utilizing the *constraint logic programming* framework. The extended constraint logic programming language, *Eclipse* [19, 20] was used as the implementation platform of the algorithm. It is well known that solving a *finite* CSP *(constraint satisfaction problems* where the variable domains are discrete and finite) is a NP-complete problem [29]. However, many of the problems that arise in practice have special properties that allow them to be solved efficiently. Indeed, in [27] it is reported (and proved) that: for any set of relations γ over a finite binary domain, i.e., $D = \{0,1\}$, where, the only operations defined are logical *AND* and *OR* then, the corresponding CSP(γ) is solvable in polynomial time. The problem translates into a *Horn-STAT* problem, which is a tractable sub-problem of the *SAT* (*satisfiability*) problem [42].

Moreover, the constraint satisfaction methods are realized by respective *network-based-consistency* techniques. With a special technique for checking network-based consistency, *restricted Generalized Forward Checking* (rGFC) [33], the CSP problem has a linear, to the number of constraints, complexity. Denote, *d*: the number of domain values; α (for *arity*): the

upper bound of the involved variables in the constraints, and *e*: the number of constraints. Then, rCFG can be implemented in time proportional to $(d+\alpha)e$.

In our case, $d = 2$, $\alpha = M$ (*M*, equals the total number of attribute- and class- values), and $e = n$ (*n*, equals the total number of cases). So, for finding a solution, MICSL needs a time proportional to $(2 + M)n$.. When a solution is found an extra constraint is added. So, the next call to the constraint satisfaction routine will need time proportional to $(2 + M)(n + 1)$ to find a solution.

Assume that *S* potential solutions are to be found. After some calculations, the total time complexity of MICSL is computed to be proportional to $S(2n+S+1)(2+M)/2$. In other words, MICSL exhibits an upper bound complexity which is proportional to the product term $M(S + n)$. It is difficult to assess the total number of potential solutions for an arbitrary domain but we may elaborate on an upper bound by the following reasonable and natural setting of the algorithm: allow at most *nM* solutions at each iteration. This upper limit is set because it is expected that, at each iteration at most, *n* cases are be covered by accommodating at most, *M* possible attribute-values in the rules. So, the upper limit of solutions is set to *mnM*, and the upper time bound to a limit proportional to $M(mnM+n)$. Therefore, the upper bound of algorithms' time complexity would be proportional to $\boldsymbol{mnM^2}$.

In the worst case, and for domains with many attributes and many attribute-values, the algorithm exhibits a more-or-less computational costly (even linear to the number of cases) performance. But this is for an upper bound and for the worst case. More careful analysis is needed in order to assess more reliable and pragmatic bounds. In future research we plan to investigate more carefully the complexity issues related with the MICSL algorithm.

4. Results on real world domains

This section presents the experimental results of the MICSL algorithm on some of the well-known and widely used real-world data sets from the UCI machine learning repository [39]. In this study we used also one additional medical domain, *Mald-Testis*, from the Pediatric Surgery Clinic of the University Hospital at Heraklion, Crete, Greece [43].

Table 1. Main characteristics of the experimental domains

Domain	#Atts	#Classes	#Data
Car	6	4	1730
Haye-Roth	4	3	264
Icu	8	2	200
Led	7	10	100
Lenses	4	3	23
Vinylon	4	8	85
Votes	16	2	435
Monks-1	6	2	432
Monks-2	6	2	432
Monks-3	6	2	432
W.Br.Cancer	6	2	680
Mald-Testis	4	3	151

The main characteristics of the data sets are reported in Table 1. The selected domains, (1) are described by nominal-valued attributes, as MICSL can manage only discrete variables (see last section for a discussion on this aspect), and (2) half of them are multi-class and are used to indicate the ability of the MICSL algorithm to cope with multi-class domains.

A *V-fold* (V = 10) *cross-validation* procedure was followed in order to evaluate the MICSL algorithm with respect to its predictive accuracy performance. In the 10-fold cross-validation method, the data set is split into ten (10) equal sized subsets. The learning algorithm is executed 10 times-- each time it is trained on all but one of the 10 subsets and then evaluated on the remaining subset [12]. The results of the 10 runs are averaged to estimate the predictive accuracy (or, the error) rate of the algorithm. The same 10-fold cross-validation procedure was performed (i.e., the same data splits were used) by using the *C4.5* decision tree induction algorithm [47], and the *CN2* [16], rule-learning algorithm.

- *MICSL & C4.5*: The reasons for comparing MICSL with C4.5 could be summarized to the following points, (1) it is publicly available, (2) it is one of the most known, extensively studied, and successfully applied machine learning algorithms, and (3) in most of the reported and published studies, C4.5 seems to produce highly accurate results, and we wanted a 'hard' comparison reference for the MICSL algorithm regarding both reliability and efficiency aspects.
- *MICSL & CN2*: Comparison results by using the *CN2* learning algorithm are also reported. CN2 and MICSL, are straightforward rule learning algorithms, as opposed to C4.5 where, decision tree's branches are interpreted as rules. So, CN2 seems a natural comparison reference.

The results of the conducted experiments are shown in Table 2.

Table 2. Experimental results: MICSL vs. C4.5u, C4.5p, and CN2

Domain	**#CS calls**	**CS system CPU***	**Total system CPU***	***C4.5u***		***C4.5p***		***CN2***		***MICSL***	
				#R	***A%***	***#R***	***A%***	***#R***	***A%***	***#R***	***A%***
Car	*202.2*	*1.50*	*6.02*	102.0	85.7	49.0	83.6	53.0	82.4	159.0	**90.4**
Hayes-Roth	*56.7*	*0.60*	*0.80*	37.0	**84.8**	33.0	79.1	27.0	82.9	21.0	84.0
Icu	*91.5*	*0.72*	*1.61*	25.5	83.2	3.0	85.0	16.0	81.3	31.5	**85.1**
Led	*66.4*	*0.06*	*0.50*	22.3	**60.9**	17.7	60.0	20.0	53.2	24.7	44.9
Lenses	*6.0*	*0.01*	*0.06*	8.0	72.2	4.0	86.1	7.0	77.8	7.8	**87.8**
Vinylon	*59.8*	*0.04*	*0.40*	59.0	**48.2**	33.0	44.8	26.0	41.0	56.0	43.5
Votes	*190.0*	*6.00*	*10.00*	194.0	93.2	7.0	94.1	18.0	**94.2**	38.0	94.0
Monks-1	*22.0*	*0.22*	*1.30*	44.7	98.2	28.7	98.4	19.0	86.8	22.0	**100.0**
Monks-2	*215.0*	*1.40*	*5.10*	227.0	46.7	1.0	**67.1**	110.0	51.9	192.0	63.6
Monks-3	*12.0*	*0.08*	*1.02*	14.0	**100.0**	14.0	**100.0**	11.0	98.4	12.0	**100.0**
W. Br.Cancer	*90.0*	*0.70*	*1.80*	87.0	94.2	10.0	93.6	26.0	90.1	98.0	**95.1**
Mald-testis	*73.0*	*0.06*	*0.52*	22.3	80.5	9.6	**85.6**	21.0	74.3	20.3	81.2

Index to the columns of Table 2

2. *#CS calls*: Represents the number of calls to the constraint satisfaction routine. This number differs from the number of induced rules because potential null solutions (corresponding rules have a null cover) may be found. In the current MICSL version, such rules are not inserted (see last section for a discussion on potential future MICSL improvements).
3. *CS system CPU*: Represents the total system CPU times (in seconds) spend during the constraint satisfaction proces.
4. *Total system CPU*: Represents the total system CPU times (in seconds) spend during the overall MICSL run.

A careful look at these figures, and at the data statistics of Table 1, indicate a close relation between the number of constraint satisfaction problems solved (i.e., #CS calls, an indicator of the number of induced rules) and the number of attributes, classes and data, for the respective domains. A close relation could be also observed between the time spend for the constraint satisfaction processes and the complexity of respective domains. More detailed analysis and discussion, on time-complexity issues regarding MICSL are presented in the last section of this paper.

5-8. The next four columns corresponds to respective results regarding the number of induced rules (#R) and predictive accuracy performance (A%) for the respective algorithms; *C4.5u* (C4.5 *unpruned*), *C4.5p* (C4.5 *pruned*), CN2 (with *star-size* set equal to 10, and *significance* of rules set equal to 0), and MICSL. For *C4.5*, the number of counted rules corresponds to the number of non-null branches in the respective trees.

Discussion

Regarding the results in Table 2 the following observations could be made. Note that figures in bold indicate the predictive accuracy superiority of the algorithm, for the reference domain.

1. MICSL outperforms CN2 in 10 out of the 12 domains. In one case, the '*votes'* domain, they exhibit nearly the same accuracy. So, comparison should be transferred between MICSL and C4.5 (both unpruned and pruned)
2. Applying a paired, one- and two-tailed t-test statistics, for MICSL against C4.5u and for MICSL against C4.5p, for all domains, *no* significant difference was observed.
3. MICSL outperforms C4.5u in 8/12 domains and C4.5p in 7/12 domains. Applying a paired, one-tail t-test statistic on the domains where, MICSL outperforms C4.5u, a significance difference was observed on the P>95% level. With a paired, two-tailed t-test statistic (a harder statistic) on the domains where MICSL outperforms C4.5p, a significance difference was also observed (on the same level, 95%). Following the inverse line of comparison (i.e., on respective domains where, C4.5u and C4.5p outperforms MICSL) no significant difference was observed.

- From the above discussion it should be evident that the presented MICSL algorithm is quite reliable with respect to its predictive accuracy performance. Especially, its comparison with C4.5 shows that MICSL exhibits at-least the same or, even higher level of performance with one of the most reliable induction methods. Here it has to be noted that, for the ' *monks-2* ' domain, C4.5 exhibits the best performance with just one rule actually, the default rule. But the default rule is not descriptive for the underlying hidden concept. Thus, MICSL should be considered as superior for this domain also.
- The significant superiority of MICSL with respect to the CN2 algorithm makes it a suitable alternative for straightforward rule-learning operations. A potential drawback is algorithm's trend to produce (induce) more rules than CN2. This could be attributed to the search heuristics of the algorithm, i.e., it searches for all the consistent models of the set of cases. Some useful hints to remedy this situation are discussed in the last section.

4. The average of the forth (4^{th}) column (i.e., the mean total system CPU cost for running MICSL over all domains) is about *2.4* sec. The same figure for CN2 is about *0.66* sec, and for C4.5 (with pruning) is about *0.2* sec. In other words, C4.5p is about twelve times, and CN2 about three times faster compared with the MICSL algorithm. This significant difference in the run-time performance of MICSL, with respect to the rival algorithms, should be attributed: (1) to the computational cost exhibited by the employed CS processes (in section 3.3 the related time-complexity issues were stated and discussed), and (2) to the computational overload imposed by the logic programming implementation platform; it is expected that a porting of the MICSL algorithm in a C language environment will improve its run-time performance (see last paragraph in the last section).

5. Related work

Interest on utilizing mathematical programming and constraint satisfaction techniques in the context of machine learning and data mining has recently increased. The main stream of approaches focuses on *discrimination analysis* (DA) tasks. DA methods explore the difference between two or more groups of cases based on one or more attributes and then, new observations are classified into the group to which they are likely to belong. The close relation of DA and classification is profound. A survey of these methods could be found in [21] and in [30, 31, 32] where, the close relation between mathematical programming and machine learning or, data mining, endeavors is also pointed out and explored.

The taxonomy of DA methods includes: (1) statistical (parametric) methods, (2) mathematical programming (MP) methods, and search methods [21]. Search methods build on recent advances in constraint satisfaction research [27, 52], and find their realization in the constraint programming and constraint logic programming framework [26]. The fundamental machinery of these methods is based on *network-based consistency* heuristics [18]. The presented MICSL algorithm is implemented in a constraint logic-programming environment, the *Eclipse* extended constraint logic programming language [19, 20]. To our knowledge there is no clear-cut constraint logic programming utilization in the context of machine learning. Therefore, this section focuses on the utilization of mathematical programming in classification and data exploration tasks. This type of data analysis, i.e., linear programming based discriminant analysis, termed *LPDA*.

LPDA methods share relative advantages over the parametric methods. First of all, are free of parametric assumptions. Moreover, complex problem formulations are easily accommodated, e.g., syntactic or semantic bias declarations imposed by domain specifics or user's requirements. The rational behind LPDA rely on the formation of specific optimization problems, and respective operations towards the discrimination between two points sets in the *n*-dimensional real space R^n (so, only 2-class problems are accommodated). In the relevant literature, LPDA methods implement stand-alone inductive procedures for: (1) induction of optimal decision trees [6], (2) clustering, by reformulating *k-Mean* and *k-Median* clustering algorithms into respective optimization problems [11], and (3) subset feature selection [7, 10].

LPDA approaches exhibit their advantages when faced with 2-class (or, binary) domains. Indeed, it is natural to expect that two-category set of points is more easily separable than multi-category ones. The limitation of the method raises from the fact that a *single-step* function optimization process is activated, trying to reduce the *miss-classification cost* of a partition. As it is reported in [6] and noted in [4], when the set of R^n points are not *'piecewise linear separable*' the method exhibits poor results. A solution is to apply LPDA iteratively and produce a hierarchy of separable sets of points, resulting in a decision tree structure. But then, the relative merits of the method, compared with the standard greedy decision tree construction methods, are missed. '*Local minima*' in the tree construction process obscure the underlying structure of the concept to be learned. In a recent study, [5], this problem is addressed and a global, non-greedy decision tree construction is proposed. The method assumes a pre-existing and known binary decision tree structure, which then tries to optimize. But prior knowledge about the application domain, and especially in the form of a tree, is not always available. So, the applicability of the method is restricted. Of course, using a standard decision-tree construction algorithm, like ID3, a tree could be generated and then optimized. But the method is applicable only to 'binary tree' structures, a strict requirement for general-purpose learning processes.

A similar with the LPDA approaches is reported in [28]. It is mainly concerned with *boolean function synthesis* problems, also restricted to 2-class problems. The rational underlying the

approach is based on the transformation of the boolean function synthesis problem into an equivalent satisfiability problem. The transformation and formulation of the respective constraints is based on logic circuit design specifics and it is not clear, at least from the reported work, how the method extends, and actually performs, on arbitrary multi-class domains. It is also unclear how the method could be extended to incorporate background knowledge and other syntactic or, semantic bias heuristics.

An interesting utilization of integer programming towards machine learning operations is made in the CLIP system [13, 14, 15]. The inductive heuristic of the approach is based on a hierarchical and iterative partition of the given set of positive cases concluding to a tree of subsets of cases. The generation of the branches is guided by the formulation of an integer programming problem the solutions of which, guarantees the discrimination of each positive case in the partition from the negative cases. By its nature the method is also restricted to 2-class domains and it is not clear how it could be generalized to multi-class problems. Furthermore, the approach does not facilitate incorporation of background knowledge.

6. Conclusions, remarks and future research

A novel concept learning algorithm, named MICSL, has been presented. It is a learning from examples algorithm, i.e., given a set of training cases it learns a set of descriptive rules. The main contribution of the algorithm comes from its uniform representation and management of both data and background knowledge and by though, it offers a flexible enough learning framework and service. The representation flexibility of MICSL rests on a method that transforms propositional cases, represented as propositional clauses, into constraint equivalents. The theoretical background as well as the validity of the transformation process are stated and studied.

The algorithm induces rules in an iterative mode, trying to identify the optimum attribute-value combinations to compose a rule. The hypotheses search heuristic of the algorithm is based on appropriately customization of linear programming and respective constraint satisfaction techniques, and follows a general-to-specific induction strategy. A constraint satisfaction problem is iteratively formed and solved. The formed CSP is composed by the set of cases' constraint equivalents and some additional constraints concerned with attribute- and class- values. The identified solutions, i.e., the labeling of the variables representing attribute and class values, are then transformed into respective rules. The process iterates by increasing a generalization parameter that specifies the required complexity of the rule. That is, the number of attribute values in the *if* part of the rule. At the end, all the minimal models of the given training set of cases are identified. These models compose the final set of induced rules. The 1-1 correspondence between the solutions of a constraint satisfaction problem and the minimal models of a training set of cases is also stated and proved.

The performance of the algorithm was tested with respect to its predictive accuracy performance on some real-world benchmark domains. Comparison results with widely used machine learning systems, such as C4.5 and CN2, are also reported. The results validate the reliability and effectiveness of the algorithm.

The setting as well as the implementation of the current version of the algorithm is just a first approach to the utilization of constraint satisfaction and mathematical programming methods in the context of concept learning. In the sequel some hints for future research, regarding algorithm's improvement, are discussed.

Enhancing MICSL. MICSL is a quite robust algorithm. The algorithm tries to find all the consistent minimal models of the data. The constraint satisfaction process triggered at each iteration, searches for solutions in a non-greedy mode, i.e., all the constraint version of cases, and all binary variables are kept active in all iterations. This unconstrained setting of the algorithm may result in two basic problems, namely: (1) overlapping rules, i.e., rules covering the same or, even identical set of cases, and (2) non-relevant or, *non-informative* attribute-values may be incorporated in the rules. In addition, there is no explicit strategy what to do when a case is classified by more than one rule or, when a case is not covered by any of the induced rules (unclassified cases). Hints for future research and development in order to overcome these problems, follows.

- *Ordering of Solutions*: The attribute-values are weighted and ordered with respect to their *information-value* in the training set of cases. The *information ratio* metric, used in the *C4.5* decision tree induction algorithm [46], is one alternative. Other metrics could be also incorporated and evaluated, see for example [34]. Then, MICSL utilizes the information-value weights in order to form a *weighted objective function* (WOF) and accommodate its respective constraint, *c(WOF)*. Denote with $w_{i;j}$, the information-value of attribute-value $A_{i;j}$ (for a domain described by *m* attributes)

  c(WOF): $\text{WOF} = \sum_{i=1}^{m} w_{i;j} \times A_{i;j}$

  Now, the constraint satisfaction routine is called to *maximize* the WOF, subject to the formed cases' constraints. The solutions with the highest WOF score will be found. That is, the *most informative* attribute-values will be utilized in the formation of the rules. Now, when a new solution is found which covers the same or, a subset of the cases already covered so far, the corresponding rule is not inserted. Thus, the cases are covered by the most informative rules. Note that, WOF should be re-computed, (a) after a new rule is inserted, and (b) considering only the uncovered cases.
- *Pruning*. The current MICSL implementation does not incorporate a pruning operation. It is in our future plans to implement and evaluate various rule-pruning techniques. Especially, the *greedy attribute-value elimination process*, based on the notion of rule's *pessimistic error rate* and adopted in the *C4.5* algorithm, [45], is a promising technique to implement.
- *Default rule*. In its current version, MICSL adopts a very simple default-rule strategy for the unclassified cases. When MICSL terminates the induced rules are executed on the training set of cases. The unclassified cases are assigned to the *most frequent* class in the training set of cases. More sophisticated default-rule strategies could be implemented and evaluated as well.

Numeric domains. In its current form, MICSL works only with nominal domains, i.e., all class- and attribute-values should be nominal. The only way to apply MICSL on numeric domains is with a pre-*discretization* of the numeric attributes. It would be an interesting, as well as tricky, research exercise to explore the extension of MICSL to *real-valued* domains. *Real interval-arithmetic* and systems for constraint satisfaction processes on real domains, like *CLP(R)*, is a promising direction to follow [25].

Large domains. The success of learning systems comes from its utilization in real-world applications. The *data-mining* endeavor is a landmark for this point. We are in the process of porting MICSL in a *C* language environment by calling a standard external linear programming routine. This will make MICSL faster, comparing with its current prolog based implementation. But the main challenge to cope with huge amounts of data still remains. In the present days, linear programs with millions of variables can be solved efficiently [9]. Moreover, parallel solutions and decomposition of large-scale mathematical programs have emerged [8]. Towards this direction, it is in our plans to investigate more efficient linear programming and constraint satisfaction techniques as an implementation platform of MICSL.

References

[1] Aha, D.W., Kibler, D., and Albert, M.K., Instance-based learning algorithms, *Machine Learning*. 6, 37-66, 1991.

[2] Banerji, R.B., *Artificial Intelligence: A Theoretical Approach*, Elsevier North Holland, NY, USA, 1980.

[3] Bell, C., Nerode, A., Ng, T., and Subrahmanian, V.S., Implementing Deductive Databases by Mixed Integer Programming, *ACM Transactions on Database Systems*. 21(2), 238-169, 1996.

[4] Bennett, K.P., and Mangasarian, O.L., Multicategory Discrimination via Linear Programming, *Optimization Methods and Software*. 3, 27-39, 1993.

[5] Bennett, K.P., Global Tree Optimization: A Non-greedy Decision Tree Algorithm. *Computing Science and Statistics*. 26, 156-160, 1994.

[6] Bennett, K.P., Blue, J., Optimal Decision Trees. *R.P.I. Math Report*. 214, 1996.

[7] Bennett, K.P., and Bredensteiner, E., Feature Minimization within Decision Trees, *Computational Optimizations and Applications*. 10(2), 111-126, 1998.

[8] Bertsekas, D.P., and Tsitsiklis, J.N., *Parallel and Distributed Computation*, Prentice- Hall, Englewood Clifs, New Jersey, 1989.

[9] Bixby, R.E., Gregory, J.W., Lustig, I.J., Marsten, R.E., and Shanno, D.F., Very large-scale linear programming: a case study combining interior point and simplex methods. *Operations Research*. 40, 885-897, 1992.

[10] Bradley, P.S., Mangasarian, O.L., and Street, W.N., *Feature selection via mathematical programming*, Technical Report 95-21, Computer Sciences Dept., University of Wisconsin, Madison, Wisconsin, 1995.

[11] Bradley, P.S., Mangasarian, O.L., and Street, W.N., Clustering via concave minimization, in *Advances in Neural Information Processing Systems 9 - NIPS* (M.C. Mozer, M.I. Jordan, and T. Petsche, eds.), Cambridge, MA, 368-374, 1997.

[12] Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J., *Classification and regression trees*, Wadsworth, Inc., Belmont, CA, 1984.

[13] Cios, K.J., and Wedding, D.K., An algorithm which learns multiple covers via integer linear programming - Part I: the CLIP2 algorithm, *Kybernetes*. 24(2), 29-50, 1995.

[14] Cios, K.J., and Liu, N., An algorithm which learns multiple covers via integer linear programming - Part II: experimental results and conclusions, *Kybernetes*. 24(3), 24-36, 1995.

[15] Cios, K.J., Wedding, D.K., and Liu, N., CLIP3: Cover learning using integer programming, *Kybernetes*. 26(5), 513-536, 1995.

[16] Clark, P., Niblett, T., The CN2 Induction Algorithm, *Machine Learning*, 3(4), 261-283 1989.

[17] Clark, P., and Matwin, S., Using Qualitative Models to Guide Inductive Learning, in *Proceedings of the 10th International Machine Learning Conference - ML93*. Kaufmann CA, 49-56, 1993.

[18] Dechter, R., and Pearl, J., Network-based heuristics for constraint-satisfaction problems, *Artificial Intelligence*. 34, 1-38, 1988.

[19] Eclipse1., *Eclipse User Manual*, ftp: // ftp.ecrc.de / pub / eclipse / pub / doc / dvi / user-manual.ps.Z, 1995.

[20] Eclipse2., *Eclipse Extensions User Manual*, ftp: // ftp.ecrc.de / pub / eclipse / pub / doc / dvi / extensions-manual.ps.Z, 1995.

[21] Erenguc, S.S., and Koeler G.J., Survey of Mathematical Programming Models and Experimental Results for Linear Discriminant Analysis, *Managerial and Decision Sciences*. 11, 215-225, 1990.

[22] Freuder, E.C., and Wallace, R.J., Partial constraint satisfaction, *Artificial Intelligence*. 58, 21-70, 1992.

[23] Fruhwirth, T., Herold, A., Kuchenhoff, V., Provost, T., Lim, P., Monfroy, E., and Wallace, M., *Constraint Logic Programming - An Informal Introduction*, ECRC Technical Report. ECRC-93-5, 1993.

[24] Giraud-Carrier, C., and Martinez, T., ILA: Combining Inductive Learning with prior Knowledge and Reasoning, *Dept. of Computer Science, University of Bristol*. Technical Report, CSTR-95-003, 1995.

[25] Jaffar, J., Michaylov, S., Stuckey, P., and Yap, R., The CLP(R) Language and System, *ACM TRans. Program. Lang. System*. 14(3), 339-395,1992.

[26] Jaffar, J., Maher, and M.J. (1994). Constraint logic programming: A survey, *J. Logic Programming*. 19/20, 503-581, 1994.

[27] Jeavons, P., Cohen, D., and Gyssens, M., Closure Properties of Constraints, *J. of the ACM - JACM*. 44(4), 527-548, 1997.

[28] Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., and Resende, M.G.C., A continuous approach to inductive inference, *Mathematical Programming*. 57, 215-238, 1992.

[29] Mackworth, A.K., Consistency in networks of relations, *Artificial Intelligence*. 8, 99-118, 1977.

[30] Mangasarian, O.L., Optimization in Machine Learning, *Computer Sciences Dept., University of Wisconsin*. Mathematical Programming Technical Report 95-01, and *SIAG/OPT Views-and-News*. 6, 3-7, 1995

[31] Mangasarian, O.L., Mathematical Programming in Machine Learning, *Computer Sciences Dept., University of Wisconsin*. Mathematical Programming Technical Report 95-06, April 1995, and in *Proceedings of Nonlinear Optimization and Applications workshop* (D. Pillo, and F. Giannessi, eds.), New York, 283-295, 1996.

[32] Mangasarian, O.L., Mathematical Programming in Data Mining, *Data Mining and Knowledge Discovery*. 1, 183-201, 1997.

[33] McAllester, D., *Constraint Satisfaction Search*, Lecture Notes for 6.824 Artificial Intelligence course, http://www.cs.unh.edu/ccc/archive/constraints/archive/dmac.ps, MIT, 1992.

[34] Mingers, J., An empirical comparison of selection measures for decision-tree induction. Machine Learning. 3/4 319--342, 1989.

[35] Mitchell, T.M., *Machine Learning*, McGraw-Hill, 1997.

[36] Morik, K., Wrobel, S., Kietz, J.-U., and Emde, W., *Knowledge Acquisition and Machine Learning - Theory, Methods, and Applications*, Academic Press, London, 1993.

[37] Muggleton, S., Feng, C.: Efficient induction of logic programs, in *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, Japan: Ohmsha, 369--381, 1990.

[38] Muggleton, S., *Inductive Logic Programming*, Academic Press, 1992.

[39] Murphy, P.M., and Aha, D.W., *UCI repository of Machine Learning Databases*, Univ. of California, Dept. of Information and Computer Science, Irvine, Calif., 1996.

[40] Nemhauser, G.L., and Wolsey, L.A., *Integer and Combinatorial Optimization*, John Wiley & Sons, Inc., NY, 1988.

[41] Ourston, D., and Mooney, R.J., Theory refinement combining analytical and empirical methods, *Artificial Intelligence*. 66, 273-309, 1994.

[42] Papadimitriou, C.H., *Computational Complexity*, Addison--Wesley, Reading, Pa., 1994.

[43] Potamias, G., Blazadonakis, M., Vassilakis, P., Gaga, L., Charissis, G., and Moustakis, V., *Case Study: The Maldescensus Testis Application*, MLT: Machine Learning Toolbox, ESPRIT P2154 project, Deliverable: MLT / WP8 / FORTH / 8.3., 1993.

[44] Quinlan, J. R., Induction of decision trees, *Machine Learning*, 1, 81-106, 1986.

[45] Quinlan, J. R., Simplifying Decision Trees, *Int. J. Man-Machine Studies*. 27, 221-234, 1987.

[46] Quinlan, J.R., Decision Trees and Multi-Valued Attributes, in *Proceedings of the Fifth International Machine Learning Conference* (J.E. Hayes, D. Michie, and J. Richards, eds.), Morgan Kaufmann, San Mateo, CA, 135-141, 1988.

[47] Quinlan, J.R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.

[48] Rossi, F., and Sperduti, A., Learning Solution Preferences in Constraint Problems, *Journal of Theoretical and Experimental Artificial Intelligence (JETAI)*. 10, 1998.

[49] Shavlik, J., Combining Symbolic and Neural Learning, *Machine Learning*, 14(3), 321-331, 1994.

[50] Tadepalli, P., and Russel, S.: Learning from Examples and Membership Queries with Structured Determinations, *Machine Learning*, 32, 245-295, 1998.

[51] Tsang, E., *Foundations of Constraint Satisfaction*, Academic Press, 1993.

[52] Van Hentenryck, and P., Saraswat., Strategic Directions in Constraint programming, *ACM Computing Surveys*. 28(4), 701-724, 1996.

[53] Winston, W.L., *Operations Research - Applications and Algorithms*, PWS-KENT Publishing Company, Boston, 1991.

[54] Yager, R.R., A mathematical programming approach to inference with the capability of implementing default rules, *Int. J. Man-Machine Studies*. 29, 685-714, 1988.