

Learning Constraints and Optimization Criteria

Samuel Kolb

KU Leuven, Leuven, Belgium
samuel.kolb@cs.kuleuven.be

Abstract

This paper presents a system that is able to automatically learn both constraints from examples and optimization criteria from rankings. Both hard and soft constraints can be learned and constraints are represented as first order logical clauses. Choosing such clauses as representations enables the use of ILP techniques such as clausal discovery. In this research, optimization criteria are represented as weighted clauses. For all examples, the clause learning system could find the essential constraints efficiently. The learning system for optimization criteria is shown to achieve accurate results and can learn criteria that identify the correct optimal solution even for few examples and noisy rankings. Finally, a procedure is outlined that allows weighted first order logic clauses to be used for optimization in practice.

Introduction

This paper introduces an approach for learning constraints and optimization criteria. Using constraints and optimization criteria such as the ones proposed in this paper allows problems to be modeled *declaratively*. This means that the representation is decoupled from the process of finding a (optimal) solution for the problem. However, non-experts often experience difficulties when modeling a problem using constraints (Puget 2004). The first goal of this research is to automate the step of modeling a problem, in order to, amongst others, make constraint solving more accessible. In this approach a few examples of what a user considers valid solutions are used by a computer system to find constraints. These constraints can then be used by an existing constraint solver.

Constraint learning attempts to find constraints automatically given examples. By using first-order logical clauses as representation for constraints, techniques from Inductive Logic Programming (ILP) can be used for learning. Hereby, this research follows the same approach as (Lallouet et al. 2010), exploiting parallels between ILP and constraint learning. In this case,

the clausal discovery algorithm (De Raedt and Dehaspe 1997) is used to find constraints.

There has been increasing interest in formulating declarative models for constraint satisfaction as well as optimization problems. By identifying soft constraints that are only satisfied by some examples and assigning weights to them based on their importance, an optimal solution can be sought. The use of weighted soft constraints to model user preferences is inspired, amongst others, by the research of (Campigotto, Passerini, and Battiti 2011). Their work examines attempts to obtain similar results in an interactive context for propositional constraints. In this research, first-order constraints are used and optimization criteria are modeled using a first-order version of weighted MAX-SAT. This paper also outlines how these constraints can be used in an existing solver to obtain an optimal solution in practice.

Since the underlying goal of this research is to facilitate the generation of a formal representation, the input format aims to be easy to use. It has been a key focus to make reasonable assumptions concerning the knowledge a user can provide. One important difference compared to similar research (including (Campigotto, Passerini, and Battiti 2011)) is that users provide partial rankings over examples instead of absolute scores.

This paper is based on research that has been conducted during the course of my masters thesis (Kolb 2015). The key contribution presented in this paper is a novel approach for learning weighted first order clauses from rankings to capture user preferences. In this context an implementation of clausal discovery has been realized that supports soft constraints. Moreover, an approach to obtaining optimal solutions for the weighted first order clauses is presented.

Background

In this section a brief overview of the important concepts and relevant literature will be provided.

Clauses First-order logical clauses consist of a disjunction of literals. Literals are defined as being either an atom (positive literal) or a negated atom (negative literal). Clauses are often expressed using a body

and a head: $head \leftarrow body$. Hereby, positive literals are grouped in the head and negative literals are grouped in the body.

Terms are used to reason about objects and consist of either constants denoting specific objects or variables that can represent any object. In this paper constants will be written in uppercase (*John*) and variables in lowercase (*x*). Relations between logical objects are represented by predicates, consisting of a predicate symbol and a set of terms (e.g. *married(John, x)*). Predicates are atoms and can be assigned a truth value.

Learning constraints Constraint learning is a hard problem. The search space of constraints is typically very large and, unlike in typical machine learning problems, there are usually only few examples to learn from.

Different systems attempt to learn constraints using different methods. The problem of having little examples can be alleviated by interactively generating data. Both the systems Conacq2 (Bessiere et al. 2007) and QuAcq (Bessiere et al. 2013) generate (partial) examples and query the user about their validity.

ModelSeeker (Beldiceanu and Simonis 2012) attempts to structure the problem variables in different ways. The system then tests for different global constraints whether they hold within this structure. This approach allows for constraint learning from a few examples.

As mentioned earlier, the research of (Lallouet et al. 2010) already tried to use ILP techniques to learn constraints. In their approach, the search space of constraints is explored in a bidirectional manner, primarily using negative examples.

Clausal Discovery The clausal discovery algorithm proposed by (De Raedt and Dehaspe 1997) attempts to learn first order logic clauses from positive examples. Using a refinement operator, the search space is traversed by incrementally generalizing clauses until they cover all examples, starting from the empty clause (\square). The result set consists of the most specific clauses covering all examples.

Logic Solver The logic system IDP (De Pooter, Wittocx, and Denecker 2013; Wittocx, Mariën, and Denecker 2008) is used to calculate whether examples satisfy clauses and is used as a theorem prover. It supports an extension of first order logic ($FO(\cdot)$) and can also be used to generate new solutions. Background knowledge is given as an IDP theory.

Optimization The Boolean Satisfiability Problem (SAT) attempts to determine whether a propositional formula can be satisfied. Formulas can be rewritten into a special form called the conjunctive normal form (CNF), which consists of a conjunction of clauses (disjunctions). An extension of this problem is the Maximum Satisfiability Problem (MAX-SAT). In MAX-SAT

the amount of satisfied clauses is maximized. By assigning positive integer weights to clauses and maximizing the sum of the weights of satisfied clauses, one obtains the weighted MAX-SAT problem. The work by (Campigotto, Passerini, and Battiti 2011) attempts to learn propositional formulas and weights automatically using examples and absolute scores. Inspired by their results, this research tries to learn first-order logic clauses and weights using examples and rankings. The resulting optimization problem can be seen as an extension of weighted MAX-SAT in which the clauses are in first-order logic.

Problem Statement

The research presented in this paper aims to learn constraints and optimization criteria based on positive examples and user preferences. Since it aims to simplify the process of generating formal representations it should be easy to use and require input that a user can reasonably be expected to provide.

Constraint learning

Given a set of examples and a limit t , the goal is to find maximally specific clauses that are satisfied by at least t of the given examples.

Examples consist of a domain and an exhaustive description of all relations that hold on the domain objects. The learned clauses are domain-independent and contain only variables.

Learning optimization criteria

Given a set of examples and a set of partial rankings over these examples, the goal is to identify soft constraints and weights learn such that the order described by these optimization criteria maximally corresponds with the given rankings.

The soft constraints consist of clauses that are satisfied by some but not all of the examples. They can be identified using a system that implements the constraint learning task. Partial rankings reason over sets of examples and describe an order over them (e.g. the ranking $e_i > e_j$ expresses the users preference of example e_i over example e_j). The weighted clauses (i.e. tuples $(weight, clause)$) can be used to calculate a score for every example:

$$score(e) = \sum_{(w, c)} w \cdot \mathbf{1}_{c \text{ covers } e}$$

Example 1 (Moving). Consider the scenario moving to a new city. The user has to choose a job, a place to live and a school. There are multiple job offers, houses and schools available across different areas of the city. User provided examples include a choice of where to work, where to live and what school is chosen. By comparing and ranking some of the examples, the user can express his preferences. Weighted soft-constraints can now be

learned and used to identify the best choice. In this case, a possible soft constraint could be that the school and work place are in the same area. The weights that have been learned indicate how desirable or how undesirable a constraint is. Since the constraints are domain independent they can be used for any (unseen) city.

Aside from learning optimization criteria, this paper presents an approach to use these optimization criteria to find an optimal solution in practice.

Approach

In this research, both constraint learning and learning optimization criteria were implemented. Clauses are learned using examples. The clause learning implementation uses the IDP system to test if clauses satisfy the examples and identify redundant clauses. When learning weighted clauses, soft constraints are first identified in the examples using the clause learning implementation. These constraints are assigned weights based on the given rankings.

Input

The input consists of global definitions and a set of examples. Definitions consist of either typing information or predicate definitions. Typing information relates directly to the problem domain, improves the accuracy and efficiency of the learned clauses and is usually easy to provide for the user. Predicate definitions define predicate symbols and the types of their terms. Two special types of predicates are allowed: calculated predicates and symmetric predicates. Calculated predicates are not included in examples, instead background knowledge is used to generate them. This background knowledge is passed directly to the IDP system. Symmetric predicates are predicates for which the order of the terms does not matter. Every example contains a domain consisting of a set of constants and an exhaustive list of the relations (predicates) that hold on these constants. If optimization criteria are to be learned, a set of pairwise over the given examples is also provided. Rankings are used instead of absolute scores since rankings only concern few examples at a time. Because rankings only ask for relative positioning of a few examples, they are easier to provide.

Example 2. Consider the moving problem (example 2). A model could, for example, use a type *Area* and predicate definitions *low_crime(Area)*, *cheap(Area)*, *school_in(Area)*, *work_in(Area)* and *live_in(Area)*. Examples would that specify various areas A_i and describe the conditions (e.g. *cheap(A₁)*, *low_crime(A₂)*) as well as the users choices (e.g. *school_in(A₂)*, *work_in(A₂)*, *live_in(A₁)*).

Clausal Discovery

The clause learning system is based on the clausal discovery algorithm (alg. 1). The logical system IDP is used for two essential functions: *covers* and *entails*.

Algorithm 1 The clausal discovery algorithm

```

 $Q \leftarrow \{\square\}, \mathcal{T} \leftarrow \{\}$ 
while  $\neg \text{isempty}(Q)$  do
   $c \leftarrow \text{next}(Q)$ 
  if  $\text{covers}(c, \text{Examples})$  then
    if  $\neg \text{entails}(\mathcal{T}, c)$  then
       $\mathcal{T} = \mathcal{T} \cup c$ 
  else
     $Q \leftarrow Q \cup \rho(c)$ 
 $\mathcal{T} \leftarrow \text{prune}(\mathcal{T})$ 
return  $\mathcal{T}$ 

```

The *covers* function calculates which examples are covered by a clause. The *entails* function calculates whether an accepted clause is logically entailed by background knowledge or clauses that are already in the result set.

Starting with the empty clause, clauses are refined until they cover enough examples. If a clause covers enough examples it is added to result set, provided it is not entailed by background knowledge or other clauses in the result set. To avoid extensive calls to *covers* and *entails*, two additional tests are used. The subset-test rejects clauses that are supersets of a clause that covers the same or all examples and has been accepted earlier. Because clauses can be formulated in multiple ways, a representative-test is used to remove clauses that are not in canonical form.

In the refinement step, clauses are extended to be more general and potentially cover more examples. The refinement operator uses a list of atoms that may be added to a clause. This list is calculated in advance, given a maximal amount of variables. Within the body and head of a clause, atoms may only be added in a specific order to avoid generating redundant clauses. Object Identity is used to specify that variables with different names must denote different objects. Additionally there are two syntactic restrictions. Clauses must be connected (i.e. new atoms must always contain a variable that has already occurred) and range-restricted (i.e. no new variables may be introduced in the head of the clause).

Optimization

The first step in finding weighted soft constraints is to identify soft constraints by using the constraint learning system with a (low) threshold. Examples are characterized by the soft constraints (clauses) which they satisfy. Therefore, every example e can be translated to a vector of boolean features by introducing a feature f_i for every clause c_i , with $f_i = \mathbf{1}_{c_i \text{ covers } e}$. Existing software can be used to learn a linear scoring function $\sum_i w_i \cdot f_i$ over these features based on the given rankings. For an unseen example, the feature vector is computed by calculating what clauses cover the new example. The scoring function can then be used to calculate a score for that example.

Table 1: Clause coverage		
Example	Covered by	Feature vector
e_1	c_1	$(1, 0, 0)$
e_2	c_2	$(0, 1, 0)$
e_3	c_1, c_2, c_3	$(1, 1, 1)$

Example 3. Consider examples e_1, e_2, e_3 , rankings $e_1 > e_2, e_2 > e_3$ and clauses c_1, c_2, c_3 . If the clauses cover examples according to table 1, then, for example, the function $(1 \cdot c_1) + (0 \cdot c_2) + (-2 \cdot c_3)$ perfectly models the given rankings. A new example that is covered by c_1 and c_2 , would be assigned a score of 1, according to this function. This score has no value in the absolute sense, it can only be used to compare it to other examples ranked by the same function. In this example clause c_1 represents a desirable property, clause c_2 is ignored because it does not influence the ranking and clause c_3 represents an undesirable property.

The SVM^{rank} software was chosen to find the scoring function since it uses a linear model and offers an efficient implementation. The weights that it assigns to the features can be used directly as weights for the optimization criteria. The input format that is used by SVM^{rank} is also supported by many other learn-to-rank implementations. Therefore, other linear ranking systems such as Coordinate Ascent (Metzler and Croft 2007) could also be used.

Optimal Solution

Solvers, such as IDP, can use clauses directly as hard constraints to generate a solution. There is no solver that can use the chosen optimization criteria directly. Weighted MAX-SAT solvers use propositional clauses and only allow for positive weights.

This optimization task can be solved in IDP, using inductive definitions, aggregates and minimization. The only limitation is that the current version only supports integer values. Therefore, the weights of the clauses are divided by the smallest absolute weight, multiplied by a constant and rounded to the closest integer if necessary.

In order to model the optimization problem in IDP, every clause c_i with variables v_1, \dots, v_n is represented by a number i . For every clause a predicate $t(i)$ is added to capture the truth value of the clause. A function $cost(i)$ specifies the cost of not satisfying the clause, which is equal to the weight of the clause.

$$t(i) \Leftrightarrow \forall v_1, \dots, v_n : c_i.$$

$$cost(i) = w_i.$$

Using t and $cost$, a function $actual(i)$ is then defined in IDP as $1 - t(i) \cdot cost(i)$. This function is used in the term $\sum_i actual(i)$ to be minimized, which will allow IDP to search for an optimal solution.

Evaluation

Several experiments aim to measure the accuracy and efficiency of the learning systems for constraints and

optimization criteria. In order to account for non-determinism, experiments measuring execution times or optimization scores are usually performed eight times and the results are averaged.

Constraints

Four problems have been used to evaluate constraint learning. The first problem is map coloring, where countries are assigned colors and neighboring countries may not have the same color. Two examples containing each three correctly colored countries are given. The second problem is sudoku and a single, solved 4×4 sudoku is given. For the third problem (elevator) three examples have been generated, two of which respect the underlying soft constraint. The last problem (co-housing) contains four hard constraints and five examples have been generated that respect the constraints.

Question 1 (Accuracy). *Are the essential constraints discovered and what influence do different parameters have on the accuracy?*

For all problems the essential constraints are found. Often additional constraints were found that describe some structure in the problems. For example, for the map coloring problem a learned constraint states that countries are never their own neighbor. These kind of constraints may help a constraint solver to work more efficiently.

The learning process is parameterized by the maximal amount of variables and literals allowed per clause. If these limits are too large, constraints are found that over-fit the training data. These constraints are too specific and will exclude valid solutions that are not in the training set. On the other hand, if the chosen limits are too small, the necessary constraints will not be found.

Over-fitting can be addressed by removing constraints that exclude valid solutions (manually or automatically) as well as providing more (or larger) training examples. Negative examples can be used to detect under-fitting, indicating that the parameter values are too small.

Question 2 (Efficiency). *How fast is the clause learning system and what is the effect of various design decisions and input on the execution time?*

Table 2 shows the execution times for several experiments. It shows that smaller problems can be solved efficiently and shows that removing the syntactical restrictions exponentially increase the search space.

All efficiency measures (i.e. symmetric predicates, subset test and representative test) have been able to improve the execution time, sometimes by more than 50%. The overhead they introduce is more than compensated by the efficiency gains they cause.

Further experiments show that increasing the number of variables or literals per clause impacts the efficiency. Especially the combination of more variables and literals can steeply increase the execution time. Therefore it would be useful to adapt these parameters dynamically.

Table 2: Execution times overview

Omitted	Problem	Average time (s)
(baseline)	Map coloring	1.581 (± 0.117)
	Sudoku	4.787 (± 0.062)
	Elevator	3.182 (± 0.073)
	Co-housing	25.903 (± 0.446)
Range restriction	Map coloring	4.629 (± 0.199)
	Sudoku	16.118 (± 0.154)
	Elevator	40.453 (± 0.319)
	Co-housing	207.768 (± 0.330)
Connected clauses	Map coloring	1.589 (± 0.110)
	Sudoku	7.068 (± 0.150)
	Elevator	6.157 (± 0.114)
	Co-housing	103.633 (± 0.131)

Adding additional examples only increases the execution time by a constant factor. Since adding more examples can help improve the accuracy, this trade-off is often worthwhile.

Question 3 (Compared to humans). *How do learned constraints compare to human programmed constraints?*

Human programmed theories for map coloring and sudoku are available on the website of the IDP system. These theories usually focus on being compact and contain only the essential constraints. Table 3 shows the results of two experiments that measure the time to find a solution for a new problem. This time is measured for the learned theories as well as for hand made theories. The learned theories are slightly adapted to be able to solve the same problems and corrected if they contain an unfair advantage.

Table 3: CPU times human vs. learned theory

Problem	Type	Average CPU time (s)
Map coloring	Human	0.968 (± 0.023)
	Learned	0.403 (± 0.015)
Sudoku	Human	1.453 (± 0.018)
	Learned	0.310 (± 0.012)

Especially for non-experts, a learning system can be useful to assist them during the modeling process. Additionally, the learning system can function in an automatic setting. These experiments show that learned constraints can be used to solve problems efficiently and even faster than hand programmed constraints for the examined cases.

Optimization

The efficiency of the learning system for optimization criteria depends mainly on the efficiency of learning soft constraints. Therefore, the experiments in this section are focused on the accuracy of the optimization criteria and the influence of different factors.

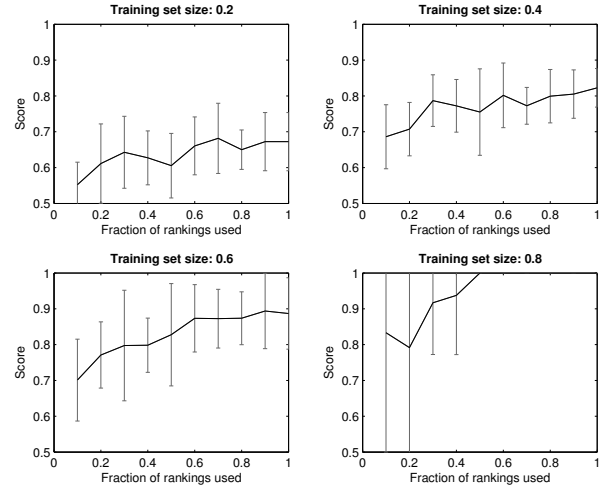


Figure 1: Influence fraction of inequalities

The moving problem with input according to example 2 is used to evaluate the learning of optimization criteria. 18 possible configurations are used as available examples. Four weighted soft constraints are used to represent the underlying model for the users preferences. Two approaches are used for evaluation. In the first approach, the examples are split into disjoint train and test sets. The second approach uses all examples as train as well as test set.

For every experiment a fraction of the training examples is selected randomly. Pairwise rankings are generated by picking two of the selected examples and predicting the better example using the underlying model. The selected examples and a random subset of the possible rankings are then used as input for learning. Noise is simulated by flipping a subset of the input rankings.

To evaluate the learned optimization criteria, all possible pairs of examples in the test set are generated. The learned model is used to predict the better example for every pair. The score is then calculated as the fraction of correctly predicted pairs. Pairs for which the underlying model ranks both example the same are omitted.

Question 4 (Accuracy). *How accurately can learned optimization criteria approximate underlying models?*

Figure 1 shows how the scores improve as the amount of examples and the fraction of included preferences increases. In all cases, more than half the pairs of examples are correctly predicted and high scores can be obtained even for small datasets. Additional experiments have shown that even for lower scores the learned optimization criteria are often capable of identifying the correct optimal solution.

The standard underlying model can be directly expressed using the soft constraints that can be learned. Even though clausal theories can be very expressive, important restrictions have been placed on the learned

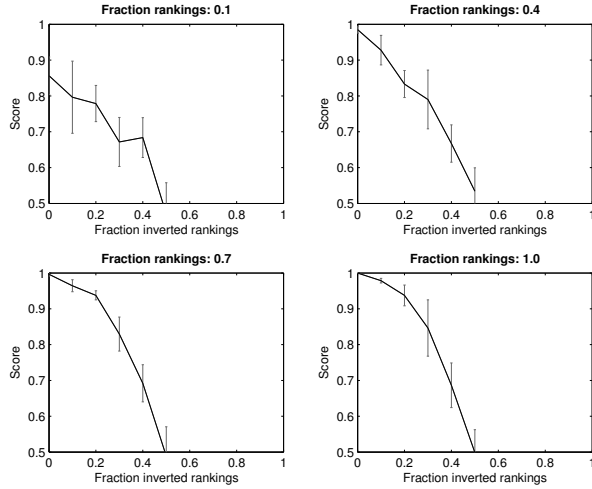


Figure 2: Influence of noise

Table 4: Scores for different thresholds (t)

$t = 1$	$t = 2$	$t = 3$	$t = 4$
0.823	0.740	0.788	0.735
(± 0.073)	(± 0.078)	(± 0.074)	(± 0.063)

clauses in this paper. In order to test the accuracy for models which cannot be directly expressed, a model consisting of two disconnected clauses has been tested as well. While there are limits to the expressibility, the learned optimization criteria were able to obtain similar scores and seem to be robust with respect to the exact formulation.

Question 5 (Noise). *What is the influence of noise on the accuracy?*

The influence of noise is shown in figure 2. Hereby, the second approach of testing is applied, using potentially overlapping train- and test sets. High scores are obtained, even despite significant levels of noise. The figure also shows that providing more rankings improves the robustness of the algorithm, even if the relative amount of noise remains unchanged.

Question 6 (Threshold). *What is the effect of the soft constraint threshold on the accuracy?*

Table 4 demonstrates that increasing the threshold used for finding soft constraints does not improve the score. This experiment used 40% of the examples as training set and 40% of the available rankings. However, if the size of the problem and examples is increased, a higher threshold will likely be appropriate.

Conclusion

The research in this thesis has focused on automatically acquiring constraints and optimization criteria from examples and rankings. Implementation for both

clause learning and clausal optimization have been provided that accomplish these tasks using first order logic clauses.

The constraint learning implementation has been able to learn the relevant hard and soft constraints in all experiments. For each problem, only a small number of examples was given to learn from. The system requires only a minimal amount of information from the user, however, it also allows for the use of expressive background knowledge. The learned constraints are domain independent, which facilitates the construction of positive examples.

Using the constraint learning implementation, the goal of learning optimization criteria has been accomplished. Optimization criteria can be learned that enable optimal solutions to be found. Even for small datasets and noisy rankings, constraints are found that enable most examples to be ranked correctly.

Aside from learning formal representations automatically from examples, this research shows how these representations can be used in practice. This also forms an important step to enable the learning of optimization criteria in an interactive setting.

Future work This research offers multiple opportunities for future work. It would be interesting to adapt the number of variables and literals in a clause dynamically. This could be accomplished, for example, by using negative examples. Learned clauses must be specific enough to not cover any negative examples.

Additionally, it would be interesting to add interactivity to the learning system for the generation of examples or rankings. Rankings expressing that examples are equally ranked are currently ignored. Whenever provided explicitly, however, it could be interesting to incorporate this information into the algorithm.

As mentioned earlier, the domain independence of the clauses has several advantages. In some cases, however, specific objects are inherently present in any problem instantiation. The constraint learning system could be enhanced to include such global constants.

Finally, it would be desirable to improve the implementation in order to tackle larger sized problems. All the experiments were conducted with problems of limited size and the computation time increases rapidly if there are more predicates, variables and literals to be used in clauses.

Acknowledgments

The author thanks his promoters Dr. Luc De Raedt and Dr. ir. Anton Dries. Furthermore, he expresses his gratitude for the help provided by Bart Bogaerts, Dr. Jesse Davis, Dr. Marc Denecker and Vladimir Dzyuba.

References

- Beldiceanu, N., and Simonis, H. 2012. A model seeker: Extracting global constraint models from positive examples. In *Principles and Practice of Constraint Programming*, 141–157. Springer.
- Bessiere, C.; Coletta, R.; O’Sullivan, B.; Paulin, M.; et al. 2007. Query-driven constraint acquisition. In *IJCAI*, 50–55.
- Bessiere, C.; Coletta, R.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Narodytska, N.; Quimper, C.-G.; and Walsh, T. 2013. Constraint acquisition via partial queries. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 475–481. AAAI Press.
- Campigotto, P.; Passerini, A.; and Battiti, R. 2011. Active learning of combinatorial features for interactive optimization. In *Learning and Intelligent Optimization*. Springer. 336–350.
- De Pooter, S.; Wittocx, J.; and Denecker, M. 2013. A prototype of a knowledge-based programming environment. In *Applications of Declarative Programming and Knowledge Management*. Springer. 279–286.
- De Raedt, L., and Dehaspe, L. 1997. Clausal discovery. *Mach. Learn.* 26(2-3):99–146.
- Kolb, S. 2015. Learning constraints and optimization criteria. Master’s thesis, KULeuven.
- Lallouet, A.; Lopez, M.; Martin, L.; and Vrain, C. 2010. On learning constraint problems. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, volume 1, 45–52. IEEE.
- Metzler, D., and Croft, W. B. 2007. Linear feature-based models for information retrieval. *Information Retrieval* 10(3):257–274.
- Puget, J.-F. 2004. Constraint programming next challenge: Simplicity of use. In Wallace, M., ed., *Principles and Practice of Constraint Programming - CP 2004*, volume 3258 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 5–8.
- Wittocx, J.; Mariën, M.; and Denecker, M. 2008. The idp system: a model expansion system for an extension of classical logic. In *Proceedings of the 2nd Workshop on Logic and Search*, 153–165.