20TH EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE
ECAI 2012

Proceedings

WORKSHOP ON COMBINING CONSTRAINT SOLVING WITH MINING AND LEARNING

CoCoMile 2012

August 27, 2012
Montpellier, France

Workshop Organization

Workshop Co-Chairs

Remi Coletta *(LIRMM, University of Montpellier, France)*
Tias Guns *(K.U. Leuven, Belgium)*
Barry O'Sullivan *(University College Cork, Ireland)*
Andrea Passerini *(University of Trento, Italy)*
Guido Tack *(Monash University, Australia)*

Program Committee

Rolf Backofen *(Albert-Ludwigs-Universität)*
Roberto Battiti *(Universitŕ degli Studi di Trento)*
Hadrien Cambazard *(University College Cork)*
Fabrizio Costa *(Albert-Ludwigs-Universität)*
Bruno Cremilleux *(Université de Caen)*
James Cussens *(University of York)*
Ian Davidson *(University of California)*
Michelangelo Diligenti *(Universitŕ degli Studi di Siena)*
Paolo Frasconi *(Universitŕ degli Studi di Firenze)*
Alan Frisch *(University of York)*
Fosca Giannotti *(ISTI - CNR)*
Emmanuel Hebrard *(LAAS-CNRS)*
Frank Hutter *(University of British Columbia)*
Matti Järvisalo *(University of Helsinki)*
Raoul Medina *(Université Blaise Pascal)*
Mirco Nanni *(ISTI - CNR)*
Peter Nightingale *(University of St Andrews)*
Siegfried Nijssen *(KULeuven)*
Jean-Marc Petit *(Université de Lyon)*
Thierry Petit *(Ecole des Mines de Nantes)*
Cédric Piette *(Université d'Artois)*
Salvatore Ruggieri *(Universitŕ degli Studi di Pisa)*
Christian Schulte *(KTH - Royal Institute of Technology)*
Meinolf Sellman *(Brown University)*
Helmut Simonis *(University College Cork)*
Peter J. Stuckey *(University of Melbourne)*
Pascal Van Hentenryck *(Brown University)*
Paolo Viappiani *(Aalborg University)*
Toby Walsh *(University of New South Wales)*
Nic Wilson *(University College Cork)*

Preface

The field of constraint solving has traditionally evolved quite independently from those of machine learning and data mining. In recent years, interest has been growing on the connections between these fields, and the potential advantages of their integration. Integration can work in two ways, on the one hand, various types of constraint solvers can be included in machine learning and data mining algorithms, for example to provide a uniform and effective way to characterize the desired solutions; on the other hand, machine learning can help in addressing constraint satisfaction problems, both at the level of search, by improving search or integrating intelligent meta-heuristics, as well as at the level of modelling, for example by learning constraints or interactively supporting a decision maker.

While promising initial results have been achieved in such directions, many options are unexplored and further research is needed in order to establish a systematic approach to this integration. The best way to reach the full potential of such integrations is in a multi-disciplinary way.

Goals The main purpose of this workshop is to provide an open environment where researchers in machine learning, data mining and constraint solving can exchange ideas and discuss promising approaches, crucial issues, open problems and interesting formalizations of new tasks. To encourage this, we will allow three different types of submissions: original contributes (unpublished work) relevant contributions recently submitted or published elsewhere (only oral) vision statements, works in progress and short overviews

Program The program includes three invited talks:

- Siegfried Nijssen *(K.U. Leuven, Belgium)*:
  Constraint programming and data mining
- Holger Hoos *(University of British Columbia, Canada)*:
  SATzilla: Portfolio-based Algorithm Selection for SAT
- Francesca Rossi *(University of Padova, Italy)*:
  Constraint-based preference modelling and elicitation

We received 14 submissions, of which 3 were rejected. The final program contains 7 papers that were accepted for oral presentation and 4 papers that were accepted as a poster.

Table of Contents

A MILP Formulation for Setwise Max-Margin Learning

Paolo Viappiani[1]

Abstract. In max-margin learning, the system aims at establishing a solution as robust as possible. In this paper we extend the idea of max-margin learning to cases where the goal is to produce a set of solutions, instead of a single one. In particular, we focus on the problem of preference elicitation, but we believe our idea could be adapted to other situations. We present a MILP formulation for our "setwise" max margin learner and discuss current ongoing works.

1 Max-margin Learning for Preference Optimization

The notion of margin is important in several machine learning algorithms. By maximizing the margin, the learner ensures robustness in the solution. In particular, a max-margin approach has been proposed for optimization with a partially specified preference profile and elicitation of further preferences [2]. Automatic assessment of preferences is an important component of many artificial intelligence systems [3].

In this paper, we show how to extend the idea of max-margin for optimizing a set and we provide a MILP formulation. The result of the computation can be used to generate a set of recommendations and to generate a "diverse" set of feasible utility functions. This can be useful for further elicitation as we may ask the user a choice query based on this set ("*Among these items, which one do you prefer ?*").

First of all, we review the standard (singleton) max-margin learning problem. We assume a multi attribute feature space $\mathbf{X}$ (we assume possible options are specified by configuration constraints) and that the user preferences are encoded by a vector of weights $\mathbf{w} = (w_1, ..., w_m)$; the utility of an option $\mathbf{y}$ is then $\mathbf{w} \cdot \mathbf{y}$.

The weight vector $\mathbf{w}$ is unknown but we are given constraints (that encode a "feasible region"). A pairwise comparison between a more preferred product $\mathbf{y}_+$ and a less preferred product $\mathbf{y}_-$ is encoded by a constraint $\mathbf{w} \cdot (\mathbf{y}_+ - \mathbf{y}_-) \geq 0$. Given a number of such pairwise comparisons, represented by a set D (the "learning set"), and, possibly, other additional constraints (representing some kind of "prior" knowledge, as for example $w_2 > 0.3$) we want to find a vector $\mathbf{w}$ of parameters such that all constraints are satisfied and does that by the largest margin.

As shown by [2] the problem can be easily formulated as a linear program. The shared margin is represented by the decision variable M, that we want to maximize. We require, for each pairwise statement in D, to have $\mathbf{w} \cdot (\mathbf{y}_+^h - \mathbf{y}_-^h) \geq M$.

$$\max_{M,\mathbf{w}} \; M$$
$$s.t. \;\; \mathbf{w} \cdot (\mathbf{y}_+^h - \mathbf{y}_-^h) \geq M \qquad \forall (\mathbf{y}_+^h, \mathbf{y}_-^h) \in D$$

The optimization can be made tolerant to contradictory or hard-to-satisfy constraints; this can be done by adding explicit slack variables ε_h.

$$\max_{M,\mathbf{w}} \; M - \alpha \sum_h \varepsilon_h \tag{1}$$
$$s.t. \;\; \mathbf{w} \cdot (\mathbf{y}_+^h - \mathbf{y}_-^h) \geq M - \varepsilon_h \qquad \forall (\mathbf{y}_+^h, \mathbf{y}_-^h) \in D \tag{2}$$

Where α is a parameter that controls the tolerance to a small number of violated constraints or constraints satisfied by a margin smaller than M. If $\alpha = 0$, then all original constraints must be strictly satisfied.

Once the optimization has been solved, the learned $\mathbf{w}$ can be used to choose a recommendation $\mathbf{x}^*$ by maximizing $\mathbf{w} \cdot \mathbf{x}$. If the space of feasible options (or products) is a configuration space encoded by linear constraints, the recommendation can be computed by solving a linear program. We refer to $\mathbf{x} \in Feasible(\mathbf{X})$ to specify that an option must satisfy the configuration constraints.

In interactive settings, one should decide the query to ask next. [2] suggests asking queries whose associated hyperplane divide the feasible region in roughly equal part, but to do so they require considering all possible pairs of options; this approach can only be feasible in small datasets and do not apply in configuration deomains. Instead, a viable method for large configuration spaces should be able to retrieve a set of options in a single optimization.

2 Setwise Max-Margin

We extend *maxmargin* optimization of preference weights to sets. Given, as before, a number of pairwise statements D and possibly additional constraints (the "learning set") our goal is to find a number of utility weight vectors $\mathbf{w}^1, ..., \mathbf{w}^k$ (with k given), consistent with out knowledge about the user, but that are representative of different parts of the weights' feasible region.

In order to do that, we require, as before, a shared margin M to hold for all constraints associated with prior knowledge and responses. We also use the same margin to enforce each vector $\mathbf{w}^i$ to be as "farther away" from each other as possible. This insight is realized by introducing a set of options $\{\mathbf{x}^1, ..., \mathbf{x}^k\}$ as decision variables (of the same cardinality k). For each i, we impose $\mathbf{x}^i$ to be the best option among them

[1] Aalborg University, Denmark, email: paolo@cs.aau.dk

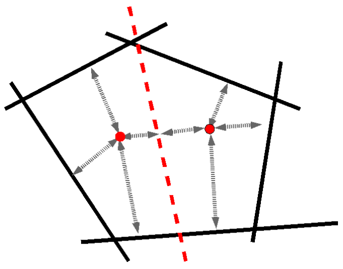


Figure 1. Pairwise max-margin ($k = 2$). The feasible region in the vector space (black constraints) is divided in two parts by the red hyperplane. The two weight vectors w_1 and w_2 (red circles), together with the hyperplane, are chosen maximize the minimum of the margins (gray arrows).

(offering higher utility) when evaluated according to $\mathbf{w}^i$ - and we require these constraints to hold by at least a margin M. The optimization is the following:

$$\max_{M,\mathbf{w}^i,\mathbf{x}^i} M - \alpha_1 \sum_h \varepsilon_h - \alpha_2 \sum_{i,j} \varepsilon'_{i,j} \tag{3}$$

$$s.t.\ \ \mathbf{w}^i \cdot (\mathbf{y}^h_+ - \mathbf{y}^h_-) \geq M - \varepsilon_h \qquad \forall (\mathbf{y}^h_+, \mathbf{y}^h_-) \in D, \forall i \in [1,k] \tag{4}$$

$$\mathbf{w}^i \cdot (\mathbf{x}^i - \mathbf{x}^j) \geq M - \varepsilon'_{i,j} \qquad \forall j \neq i;\ i,j \in [1,k] \tag{5}$$

$$\mathbf{x}^i \in Feasible(\mathbf{X}) \qquad \forall i \in [1,k]$$

Note that, we are choosing the options $\mathbf{x}^1, ..., \mathbf{x}^k$ and the weights $\mathbf{w}^1, ..., \mathbf{w}^k$ simultaneously: since we want to maximize M, the optimizer will be better off by choosing a set of outcomes $\mathbf{x}^i$ that divide the weight space roughly equally, and the utility functions such each $\mathbf{w}^i$ should lie (intuitively) near the centre of each subregion.

This initial formulation is problematic, as we have quadratic terms. However, there is a solution: in fact, by using integer programming tricks, the problem can be formulated as a mixed integer linear program (MILP).

$$\max\ M - \alpha_1 \sum_h \varepsilon_h - \alpha_2 \sum_{i,j} \varepsilon'_{i,j} \tag{6}$$

$$s.t.\ \ \mathbf{w}^i \cdot (\mathbf{y}^h_+ - \mathbf{y}^h_-) \geq M - \varepsilon_h \qquad \forall (\mathbf{y}^h_+, \mathbf{y}^h_-) \in D, \forall i \in [1,k] \tag{7}$$

$$\sum_z A^i_z - B^{i,j}_z \geq M - \varepsilon'_{i,j} \qquad \forall j \neq i;\ i,j \in [1,k] \tag{8}$$

$$A^i_z \leq w{\uparrow}\, x^i_z \qquad \forall i \in [1,k], z \in [1,m] \tag{9}$$

$$A^i_z \leq w^i_z \qquad \forall i \in [1,k], z \in [1,m] \tag{10}$$

$$B^{i,j}_z \geq w^i_z - C \cdot (1 - x^j_z) \qquad \forall j \neq i \in [1,k], z \in [1,m] \tag{11}$$

$$B^{i,j}_z \geq 0 \qquad \forall j \neq i \in [1,k], z \in [1,m] \tag{12}$$

$$\mathbf{x} \in Feasible(\mathbf{X}) \qquad \forall i \in [1,k]$$

In the optimization, the decision variables are the following:

- M is the size of the shared margin
- $\mathbf{w}^1, ..., \mathbf{w}^k$ is a set of utility vectors; each vector defined over m attributes (features): $\mathbf{w}^i = (w^i_1, ..., w^i_m)$
- $\mathbf{x}^1, ..., \mathbf{x}^k$ is a set of configurations (options) with each configuration $\mathbf{x}^i = (x^i_1, ..., x^i_m)$; each element x^i_z is binary
- ε slack variables to represent cost of unsatisfied constraints in D; ε' slack variables to represent cost of not respecting the margin M when choosing the hyperplanes
- $\mathbf{A}^i$ encodes the vector $(w^i_1 x^i_1, ..., w^i_m x^i_m)$, the element-by-element product of $\mathbf{w}^i$ and $\mathbf{x}^i$: $A^i_z = w^i_z x^i_z$
- $\mathbf{B}^{i,j}$ encodes $(w^i_1 x^j_1, ..., w^i_m x^j_m)$, the element-by-element product of $\mathbf{w}^i$ and $\mathbf{x}^j$

In addition to the decision variables, we make use of the following parameters:

- k is the size of the set
- α_1, α_2 control the tolerance with respect to violated constraints (we can differentiate the tolerance for the two different types of constraints).
- D is the *learning set*: a set of pairwise comparisons known to the system
- $w{\uparrow}$ is any upper bound of the value of the utility weights
- C is an arbitrary large number
- $\mathbf{y}^h_+, \mathbf{y}^h_-$ are pairs of configurations (options) in the learning set D, with $\mathbf{y}^h_+$ preferred to $\mathbf{y}^h_-$

The solver aims at setting M, that is a decision variable, as large as possible. Constraint 5 of the original program is replaced by constraint 8, enforcing M to be smaller than $\sum_z A^i_z - B^{i,j}_z$ for any i, j. In order to maximize M, the solver will try to keep the A^i_z as large as possible and the $B^{i,j}_z$ as small as possible. The fact that $\sum_z A^i_z - B^{i,j}_z$ evaluates to $\mathbf{w}^i \cdot (\mathbf{x}^i - \mathbf{x}^j)$ is achieved by setting additional constraints. Constraints 9 and 10 together force each $\mathbf{A}^i$ to be the element-by-element product of $\mathbf{w}^i$ and $\mathbf{x}^i$ ($A^i_z = w^i_z x^i_z$): if $x^i_z = 0$ then (constraint 9) also A^i_z must be 0; otherwise (if $x^i_z = 1$) $A^i_z \leq w^i_z$ (constraint 10) but because of the objective function maximizing M, and each A constrain M, the solver will set A^i_z to w^i_z. Similarly constraints 11 and 12 together make it so that $\mathbf{B}^{i,j}$ encodes $(w^i_1 x^j_1, ..., w^i_m x^j_m)$, the element-by-element product of $\mathbf{w}^i$ and $\mathbf{x}^j$ (as C is an arbitrary large constant, constraint 11 is binding only when x^j_z is 1, otherwise it is always satisfied).

3 Discussion

This paper describes an extension of maximum margin learner to sets. We are currently planning to perform experiments to evaluate the efficiency of this approach. We will compare to baseline methods, maximizing product diversity for a single learned weight vector. We are interested in computation time, but also in the quality of the elicitation using our method in an interactive setting. In particular, we are interested in comparing this approach for generating recommendation sets with others based on Bayesian inference [5] and minimax regret [4]. We are also interested in learning approaches ensuring sparsity in the parameter space [1].

REFERENCES

[1] Paolo Campigotto, Andrea Passerini, and Roberto Battiti. Active learning of combinatorial features for interactive optimization. In Carlos A. Coello Coello, editor, *LION*, volume 6683 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2011.

[2] Krzysztof Gajos and Daniel S. Weld. Preference elicitation for interface optimization. In Patrick Baudisch, Mary Czerwinski, and Dan R. Olsen, editors, *UIST*, pages 173–182. ACM, 2005.

[3] Bart Peintner, Paolo Viappiani, and Neil Yorke-Smith. Preferences in interactive systems: technical challenges and case studies. *AI Magazine*, 29(4):13–24, 2008.

[4] Paolo Viappiani and Craig Boutilier. Regret-based optimal recommendation sets in conversational recommender systems. In *Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys09)*, pages 101–108, New York, 2009.

[5] Paolo Viappiani and Craig Boutilier. Optimal bayesian recommendation sets and myopically optimal choice query sets. In *Advances in Neural Information Processing Systems 23 (NIPS)*, Vancouver, 2010.

Algorithm Configuration for Portfolio-based Parallel SAT-Solving

Holger Hoos [1] and **Kevin Leyton-Brown** [1] and **Torsten Schaub** [2] and **Marius Schneider** [2]

Abstract. Since 2004, the increases in processing power enabled by Moore's law have been primarily achieved by means of multi-core processor architectures. To make effective use of modern hardware when solving hard computational problems, it is therefore necessary to employ parallel solution strategies. In this work, we demonstrate how effective parallel solvers for SAT, one of the most widely studied NP-complete problems, can be produced automatically from any existing sequential, highly parametric SAT solver. Our approach uses an automatic algorithm configurator to produce a set of configurations to be executed in parallel. Applied to the state-of-the-art SAT solver *Lingeling*, our fully automated procedure produced 4-core solvers with speedups of up to 2.79-fold on a diverse set of instances from the *application* category of the 2003–2011 SAT Competitions. Our best automatically generated parallel portfolio of *Lingeling* configurations outperforms *Plingeling*, the gold medal winner of the application track (wallclock time) of the 2011 SAT Competition, and *ManySAT*, the winner of the special prize for parallel solvers for application instances of the 2009 SAT Competition. We furthermore demonstrate that, when applied to the state-of-the-art multi-threaded SAT and ASP solver *clasp*, our automated approach yields parallelization speedups matching those achieved through the considerable efforts of a human expert with extensive knowledge of the solver.

1 Introduction

Over most of the last decade, additional computational power has come primarily in the form of increased parallelism. As a consequence, effective parallel solvers are increasingly key to solving computationally challenging problems. Unfortunately, the manual construction of parallel solvers is nontrivial, often requiring fundamental redesign of existing, sequential approaches. It is thus very appealing to identify generic methods for the construction of parallel solvers from inherently sequential sources. Indeed, the prospect of a substantial reduction in human development cost means that such approaches can be impactful, even if their results are performance does not reach that of special-purpose parallel designs—just as high-level programming languages are useful, even though compiled software tends to fall short of the performance that can be obtained from expert-level programming in assembly language. One promising approach for parallelizing sequential algorithms is the design of parallel algorithm portfolios [15, 8].

In this work, we study generic methods for generating parallel portfolios from a single, highly parametric sequential solver design for a given problem. As such, it can be understood as an instance of the programming by optimization paradigm [13], providing concrete software tools that leverage algorithm configurators and a user-specified design space to substitute for human development effort. In particular, unlike other approaches (further discussed in Section 2), our methods do not depend on the availability of multiple complementary solver designs. We evaluate our methods in the SAT domain, which we chose because it is widely studied and very relevant to academia and industry. We thus have access to well-known state-of-the-art highly parametric solvers, and are assured that the bar for demonstrating efficacy of parallelization strategies is appropriately high.

We consider two scenarios. In the first, there is no communication between component solvers, and the parallel portfolio can be generated fully automatically from a single, sequential parametric solver. In this case, the design space for a parallel portfolio of size k corresponds to the kth Cartesian power of the design space of the given sequential solver. To evaluate our methods in this setting, we chose *Lingeling*, a prominent, highly parametric state-of-the-art SAT solver underlying the parallel solver that won a gold medal in the application (wall-clock) track of the 2011 SAT Competition.

Our second scenario allows for communication between component solvers in a parallel portfolio. Here, component solvers are copies of a single, parametric sequential solver that communicate through a simple mechanism; for example, in SAT, they might share learned clauses (see, e.g., [10].) The communication mechanism is problem-specific and designed by a human expert, resulting in the same design space as in our first scenario, augmented to further include design choices that span the component solvers (the communication mechanism itself, preprocessing strategies, etc). To evaluate our methods in this setting, we chose to study the state-of-the-art, highly parametric, multi-threaded SAT and ASP solver *clasp*.

The key idea underlying our approach for handling both scenarios lies in the use of automated algorithm configurators, which are now quite mature and have been demonstrated to achieve impressive performance improvements for different solvers on many problems (see, e.g., [18, 1, 28, 21, 16, 17]). The configuration spaces arising in the context considered here are very large and therefore present a considerable challenge even to the best configurators. Therefore, in addition to a rather straightforward approach in which all components of a given parallel portfolio are configured simultaneously, we introduce a greedy approach that adds one component solver at a time. Our results demonstrate that this second approach works particularly well and produces parallel portfolios whose performance on standard 4-core CPUs compares favourably with that of well-known, hand-crafted parallel SAT solvers.

[1] Department of Computer Science, University of British Columbia, Vancouver, BC (Canada), {hoos,kevinlb}@cs.ubc.ca

[2] Institute of Computer Science, University of Potsdam, Germany, {torsten,manju}@cs.uni-potsdam.de

2 Related Work

Well before the advent of the current trend towards multi-core computing, the potential benefits of parallel algorithm portfolios were identified in seminal work by Huberman et al. [15]. Gomes & Selman [8] further investigated conditions under which such portfolios outperform their constituent solvers. Both lines of work considered prominent constraint programming problems (graph colouring and quasigroup completion), but neither presented methods for automatically constructing portfolio solvers. More recently, such methods have been introduced for parallel portfolios in which the allocation of computational resources to algorithms in the portfolio is static [23, 29], as well as in cases where it can change over time [6]. All of these methods build a portfolio from a relatively small candidate set of distinct algorithms. While in principle, these methods could also be applied given a set of algorithms expressed implicitly as the configurations of one parametric solver, in practice, they are useful only when the set of candidates is relatively small. The same limitation applies to existing approaches that combine algorithm selection and scheduling, notably *CPHydra* [22], which also relies on cheaply computable features of the problem instances to be solved and selects multiple solvers to be run in parallel.

In contrast, our work is concerned with building parallel portfolios from very large sets of candidate algorithms which are expressed as parameter settings of high-performance solvers such as *Lingeling* and *clasp*. Our approach critically relies on the availability of an effective algorithm configurator, such as *ParamILS* [19, 18], *GGA* [1], or *SMAC* [17, 20]. It is conceptually related to the *Hydra* and *ISAC* procedures for constructing portfolio-based algorithm selectors [28, 21]. Like these methods, our approach uses an algorithm configurator to determine a set of configurations that complement each other well. Furthermore, like *Hydra*, our GREEDY portfolio construction procedure relies on the idea of determining such configurations one at a time, to achieve a maximum incremental performance improvement in each iteration. However, both *Hydra* and *ISAC* construct per-instance algorithm *selectors*: they do not run multiple solvers in parallel, but instead select a single solver based on instance features. To our knowledge, our work is the first to show how to automatically construct effective parallel portfolios from single, highly parametric solvers.

Another conceptually related approach is *aspeed* [14], which computes (parallel) algorithm schedules by taking advantage of the modeling and solving capacities of Answer Set Programming. Unlike our approach, *aspeed* is based on a diverse set of solvers and does not use an algorithm configurator to optimize its configurations.

Parallel SAT solvers have received increasing attention in recent years. *ManySAT* [10, 11, 9] was one of the first parallel SAT solvers. It is a static portfolio solver that uses clause sharing between its components, each of which is a manually configured, DPLL-type SAT solver based on *MiniSat* [5]. *Plingeling* [3, 4] is based on a similar design; its recent version 587, which won a gold medal in the application track of the 2011 SAT Competition (wrt. wall clock time on SAT+UNSAT instances), shares unit clauses as well as equivalences between its component solvers. Similarly, *CryptoMiniSat* [26], which won silver in the application track of the 2011 SAT Competition, shares unit and binary clauses. *clasp* [7] is a state-of-the-art solver for SAT and ASP [2] that supports parallel multithreading (since version 2.0.0) for search space splitting and/or competing strategies, both combinable with a portfolio approach. *clasp* shares unary, binary and ternary clauses, and (optionally) offers a parameterized mechanism for distributing and integrating (longer) clauses. Finally, *ppfolio* [24] is a simple, static parallel portfolio solver for SAT without clause sharing that uses *CryptoMiniSat*, *Lingeling*, *clasp*, *TNM* [27] and *march_hi* [12] in their default configurations as component solvers and won numerous medals in the 2011 SAT Competition. Like the previously mentioned portfolio solvers for SAT, *ppfolio* was constructed manually, but uses a very diverse set of high-performance solvers as its components. Overall, our approach can be understood as an automatic method for replicating the (hand-tuned) success of solvers like *ManySAT*, *Plingeling*, *CryptoMiniSat* or *clasp*, which are inherently based on different configurations of a single parametric solver.

Algorithm 1: Portfolio Configuration Procedure GLOBAL

```
Input  : parametric solver $A$ with configuration space $C$;
         configuration space $C_g$ for communication mechanism
         between component solvers; desired number $k$ of
         component solvers; instance set $I$; performance metric
         $m$; configurator $AC$; number $n$ of independent
         configurator runs; total configuration time $t$
Output : parallel portfolio solver $A^k$
1 for $j := 1..n$ do
2     obtain configuration $c_j$ by running $AC$ on $A^k$ with
      configuration space $C^k \times C_g$ on $I$ using $m$ for time $t/n$
3 choose $\hat{c} \in \{c_1, \dots, c_n\}$ for which $A^k$ gives optimal
  performance on $I$ according to $m$ return $\hat{c}$
```

3 Parallel Portfolio Configuration

We now describe two new methods for generating a parallel solver portfolio from a single parametric solver, A, with configuration space C. We call the given set of problem instances I; our goal is to obtain optimized performance according to a given metric m. (In our experiments, we minimize PAR10 [18].) We use $A^k = [A_1, \dots, A_k]$ to denote a parallel portfolio with k component solvers, each of which is a configuration of A. The configuration space of A^k is $C^k = \prod_{i=1}^{k} C$ in the case where there is no communication between the component solvers (apart from coordinated launch and termination), and $C^k \times C_g$ in the case where $A_1, \dots, A_k$ share information throughout a run, where C_g is the set of all possible settings of the parameters of the communication mechanism and any other global logic. Let AC denote a generic algorithm configuration procedure (in our experiments, we used *ParamILS* [19, 18]). Following our standard practice (see e.g., [20]) we perform multiple independent runs of AC, keeping the configuration with the best performance on I. We model the case of non-communicating component solvers as $C_g := \{\emptyset\}$.

Simultaneous configuration of all component solvers (GLOBAL). Our first portfolio configuration method is the straightforward extension of standard algorithm configuration to the construction of a parallel portfolio (see Algorithm 1). Specifically, if A has ℓ parameters, we treat the portfolio A^k as a single algorithm with ℓ^k parameters and configure it directly. As noted above, we perform n parallel runs of AC. These runs can be performed in parallel, meaning that this procedure requires wall clock time of t/n if n cores are available. Nevertheless, the practicality of this approach is limited by the fact that the global configuration space $C^k \times C_g$ to which AC is applied grows exponentially with k. However, given a powerful configurator, a moderate value of k and a reasonably sized C (and C_g), this simple approach could be quite effective, especially when compared to the manual construction of a parallel portfolio.

Algorithm 2: Portfolio Configuration Procedure GREEDY

```
Input  : parametric solver $A$ with configuration space $C$;
         configuration space $C_g$ for communication mechanism
         between component solvers; desired number $k$ of
         component solvers; instance set $I$; performance metric
         $m$; configurator $AC$; number $n$ of independent
         configurator runs; total configuration time $t$
Output : parallel portfolio solver $A^k$
1 $A^0$ := [empty portfolio]
2 for $i$ := $1..k$ do
3     for $j$ := $1..n$ do
4         obtain configuration $c_j^i$ by running $AC$ on
          $A^i := A^{i-1}||A$ with configuration space
          $\prod_{l=1}^{i-1}\{\hat{c}^l\} \times C \times C_g$ on $I$ using $m$ for time $t/(k \cdot n)$,
          where $A^{i-1}||A$ denotes the portfolio obtained by
          extending $A^{i-1}$ with algorithm $A$
5     let $\hat{c}^i \in \{c_1^i, \ldots, c_n^i\}$ be the configuration for which $A^i$
      achieved best performance on $I$ according to $m$, and let $\hat{c}_i$ be
      the configuration of the component solver most recently
      added to $A^i$
6 return $\hat{c}^k$
```

Iterative configuration of component solvers (GREEDY). For use in what we expect to be the typical case where $C^k \times C_g$ is too large to be effectively searched by AC, we introduce an iterative procedure that adds and configures component solvers one at a time (see Algorithm 2). The key idea is to use AC only to configure the component solver added in the given iteration (and the communication mechanism, as applicable, once there are two or more components), leaving all other components clamped to the configurations that were determined for them in previous iterations. The procedure is greedy in the sense that in each iteration i, it attempts to add a component solver to the given portfolio A^{i-1} in a way that myopically maximizes the performance of the new portfolio A^i (Line 4). Obviously, for $k > 1$, even if we assume that AC finds optimal configurations in each iteration, this greedy procedure is not guaranteed to find a globally optimal configuration of the entire portfolio. However, the configuration tasks in each iteration are much easier than those performed by GLOBAL for even moderately sized portfolio, giving us reason to hope that under realistic conditions, GREEDY might perform better than GLOBAL, especially for large configuration spaces C and C_g, and for comparatively modest time budgets t. Finally, notice that this procedure only runs portfolios of size i in each iteration i; therefore, if there is a cost to computing cycles for each parallel CPU or CPU core, there are savings in earlier iterations $i < k$. (However, note that unlike *Hydra*, which GREEDY resembles, we do run entire portfolios in each iteration rather than individual solvers.) Observe that while the sets of n independent configurator runs in Line 4 can be performed in parallel (as in GLOBAL), the choice of the best-performing configuration $\hat{c}^i$ has to be made after each iteration i, introducing a modest overhead compared to the cost of the actual configuration runs.

4 Experiments

To empirically evaluate our approach for creating and optimizing parallel algorithm portfolios, we applied our GLOBAL and GREEDY methods to two state-of-the-art SAT solvers: *Lingeling* and *clasp*. Specifically, we compared performance-optimized sequential and parallel versions of both solvers to our GREEDY method, running on four cores. Finally, we assessed the performance of the parallel solvers obtained using our approach relative to other parallel SAT solvers. A more detailed description of our experimental findings is available at `http://www.cs.uni-potsdam.de/parfolio`.

Scenarios. We compared six experimental scenarios for each solver. We use the terminology *Default-SP* to denote a single-processor solver's default configuration, and analogously *Default-MP4* for an out-of-the-box four-processor version. We contrasted these solver versions with three versions obtained using automated configuration: *Configured-SP*denotes the best (single-processor) configuration obtained from 40 independent configurator runs on a training set, while *Global-MP4* and *Greedy-MP4* represent the portfolios obtained using our methods from Section 3 for $n = 10$ and $k = 4$.

Solvers. We applied our approach to the two highly parameterized, state-of-the-art SAT solvers *Lingeling* version 276 [3] and *clasp* version 2.0.2 [7]. *Lingeling* has 58 parameters, which (after discretization) gave rise to a configuration space of size about 10^{45}. Our parallel portfolio version of *Lingeling* was implemented based on a simple script that runs a given number of *Lingeling* instances independently in parallel and without communication ($C_g := \{\emptyset\}$). We did not apply our methods to *Plingeling*, the 'official' parallel version of *Plingeling*, because it lacks configurable parameters. However, we did compare our methods to *Plingeling*. (Single-processor) *clasp* has 25 parameters, which—discretized by the developer—induce a space of about 10^{13} configurations. *clasp* comes with a native multi-threaded architecture, in which each parallel thread can be configured nearly as flexibly as the sequential solver. Preprocessing is controlled and configured (C_g) globally for all threads. We did not consider active clause sharing in our experiments, but multi-threaded *clasp* passively shares unary, binary, and ternary clauses. Overall, four-threaded *clasp* can be configured in about 10^{53} distinct ways. *clasp*'s default configurations were determined by its main developer with considerable manual effort; the default parallel portfolio version of *clasp*, *Default-MP4*, was entered in the 2011 SAT Competition.

Instance Sets. We conducted our experiments on instances from the *application (industrial)* categories of the 2003–2011 SAT Competitions. Our configuration experiments distinguish a training and a test set. We used the same training set as Schneider and Hoos [25], consisting of 276 instances from the 2003–2009 SAT Competitions. Our test set was comprised of all application (industrial) instances used in the 2003 and 2011 SAT Competitions, with the exception of instances already included in our training set: 679 instances overall. We chose a captime of 600 seconds for solver runs on training instances performed during configuration, and a captime of 5000 seconds (as in the 2011 SAT Competition) when evaluating solvers on the test set.

Evaluation Criteria. All solvers were configured and evaluated based on PAR10 scores [18], which treat timed-out runs as having taken 10 times the captime. We compared solvers using three measures. First, *overall speedup* measures the speedup in terms of total PAR10 scores, disregarding instances from each table in what follows that were not solved by any solver. Second, *(arithmetic) average speedup* takes the average over the set of the compared solvers' speedups, considering only instances that could be solved by both compared solvers. (We note that this measure was previously used both in the 2008 SAT Race and by Hamadi et al. [11]; however, if there

	PAR10	Overall Speedup *vs* Default-SP	Overall Speedup *vs* Configured-SP	Avg. Speedup *vs* Configured-SP	Geo. Avg. Speedup *vs* Configured-SP
Default-SP	3747	1.00	0.93	1.44	0.98
Configured-SP	3499	1.07	1.00	1.00	1.00
Plingeling	3066	1.22	1.14	7.39	1.46
Global-MP4	2734	1.37	1.27	10.47	1.36
Greedy-MP4	1341	2.79	2.61	3.52	1.60

Table 1: PAR10 scores and speedups on application/industrial SAT instances achieved by single-processor (SP) and 4-processor (MP4) versions of *Lingeling*.

	PAR10	Overall Speedup *vs* Default-SP	Overall Speedup *vs* Configured-SP	Avg. Speedup *vs* Configured-SP	Geo. Avg. Speedup *vs* Configured-SP
Default-SP	7560	1.00	0.82	4.46	1.04
Configured-SP	6170	1.23	1.00	1.00	1.00
Default-MP4	2324	3.25	2.65	7.58	2.15
Global-MP4	3604	2.10	1.71	6.36	1.44
Greedy-MP4	2277	3.32	2.71	9.47	2.14

Table 2: PAR10 scores and speedups on application/industrial SAT instances achieved by single-processor (SP) and 4-processor (MP4) versions of *clasp*.

are instances solved by only one solver, disregarding these when measuring speedup can bias results against that solver.) Finally, *geometric average speedup* takes the nth root of the product of the elements of the set of the compared solvers' speedups over the default, again considering only instances that could be solved by both compared solvers.

We now compare the three measures. The overall speedup assesses the speedup obtained in a situation where a stream of problem instances has to be solved, and our test set is representative of that stream. This is the measure we favour, because performance on hard instances is often the most important, because this measure is much less sensitive to outliers, and because it does not require dropping instances that are solved only by the single, best-performing solver. Thus, while we include the other measures in our tables, we do not discuss them in the text in what follows. Average and geometric average speedups are nevertheless also useful for considering situations where there is substantial uncertainty over the difficulty of instances that will ultimately be faced, and therefore consistent speedups across the entire training set (rather than just hard instances in that set) is important. We note that, unlike geometric average speedup, average speedup can give rise to situations where algorithms A and B have speedups > 1 of A against B *and* B against A simultaneously. (To see this, consider running times $1, 2$ for A and $2, 1$ for B on two given instances.)

We performed all solver and configurator runs on Dell PowerEdge R610 systems with an Intel Xeon E5520 CPU with four cores (2.26GHz), 48GB RAM running 64-bit Scientific Linux.

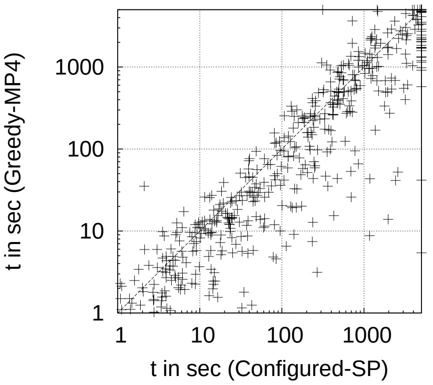
Configuration Experiments. We used the FocusedILS variant of *ParamILS* (version 2.3.5) [18], one of the best algorithm configurators currently available. To enable fair performance comparisons, in the case of *Configured-SP* and *Global-MP4* we allowed 8 CPU days of configuration time and 1 CPU day for validation runs per configurator run, which amounted to a total of 360 CPU days. (Validation runs were used to choose the best among a set of configurations; they relied on the same training set as the configuration runs. The test set was used only for evaluating the different methods.) For *Greedy-MP4*, we allowed for 2 CPU days of configuration time and 1 CPU days of validation time per configurator run, which amounted to a total of about 300 CPU days for $k = 4$. When using a cluster of dedicated machines with 4-core CPUs, each of those solver versions could be produced within 9 days of wall-clock time.

	PAR1	PAR10	Timeouts
Virtual Best Solver	1334	10480	138
ppfolio	1646	13310	176
Greedy-MP4 (*Lingeling*)	1717	13712	181
Plingeling (587)	1684	13812	183
Greedy-MP4 (*clasp*)	1856	15310	203
clasp (MT)	1837	15357	204
Plingeling (276)	1850	15437	205
ManySAT(1.1)	1887	16003	213
ManySAT(2.0)	1998	17373	232

Table 3: Comparison of our best parallelization approach, GREEDY, with other parallel SAT solvers from the 2011 SAT Competition in the four-processor setting. (The performance of the Virtual Best Solver is the minimal runtime of each instance given a portfolio of solvers.)

Parallelization speedups. Table 1 presents the results of our experiments with *Lingeling* in the communication-free scenario. We observe that single-processor configuration offered very little benefit here, with only small improvements in PAR10 score. Somewhat better results were obtained for *Plingeling*, but despite access to four cores only achieved an overall speedup of 1.22 as compared to the *Lingeling* default. Our *Global-MP4* method outperformed *Plingeling*, but only slightly, achieving an overall speedup of 1.37 times the default. Our *Greedy-MP4* method made the best use of its four cores, achieving a 2.79-fold speedup (see also Figure 1a). Using a permutation test (10 000 iterations, $p = 0.05$), we confirmed that *Greedy-MP4*'s performance significantly exceeded that of the other methods.

Table 2 summarizes the results of our experiments with *clasp*. Here again we observe small gains from configuring the single-processor solver, and *Greedy-MP4* outperforming *Global-MP4*. Overall, *Greedy-*



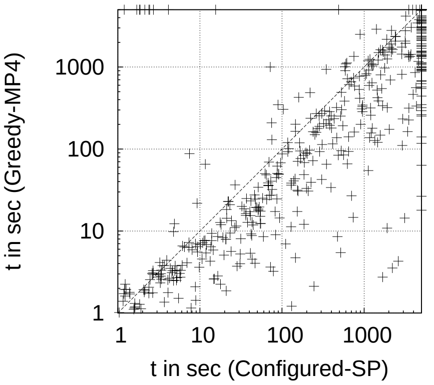


Figure 1: Performance of *Greedy-MP4* vs *Configured-SP* for *Lingeling* (left) and *clasp* (right); each cross represents one SAT instance from our evaluation set.

MP4 performed even better in this domain, achieving a total speedup of 3.32 over the single-processor default (see also Figure 1b). *Greedy-MP4* achieved slightly (but not significantly) better performance as compared to *clasp*'s multi-processor default. However, this default was developed through extensive human effort and (as a SAT Competition entrant) had previously targeted the same data we used to evaluate it. Thus, we see our automated methods' ability to match *Default-MP4*'s performance as an encouraging finding.

Comparison to other parallel solvers. Finally, Table 3 presents a comparison of our methods' performance relative to other 4-processor parallel solvers. We note a few interesting points here. First, *Plingeling*, the 2011 SAT Competition gold medal winner in the application multi-core track, appears only in 3rd place; however, we also note that the competition used 8 processor cores. Second, our *Greedy-MP4* (*Lingeling*), which is based on version 276 of *Lingeling* from 2010, performed as well as the new *Plingeling*, version 587. Third, although the ASP-solver *clasp* was designed for SAT instances more similar to those from the competition's crafted (rather than application) track, *clasp* (in both its default and our *Greedy-MP4* variants) solved more instances than both *ManySAT* versions and slightly more than *Plingeling*, version 276. Fourth, we note that *ManySAT*'s performance was weaker than one might expect given the speedups described in [11]; however, these results were based on (arithmetic average) speedups over the single-processor variant of *ManySAT* rather than *MiniSat* 2.1 (confirmed through personal communication with the authors). Finally, *ppfolio*'s strong performance indicates that portfolios of complementary solvers can yield even stronger performance than parallel portfolios constructed from single parameterized solvers. This is further confirmed by the excellent performance of the perfect per-instance solver selector over the solvers we considered ("virtual best solver"). We aim to consider automatically constructed parallel portfolios that span multiple parametric solvers in future work.

Acknowledgments

T. Schaub and M. Schneider were supported by the DFG projects under SCHA 550/8-1/2 and SCHA 550/9-1.

REFERENCES

[1] C. Ansótegui, M. Sellmann, and K. Tierney, 'A gender-based genetic algorithm for the automatic configuration of algorithms', in *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09)*, volume 5732 of *Lecture Notes in Computer Science*, pp. 142–157. Springer-Verlag, (2009).

[2] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press, 2003.

[3] A. Biere, 'Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010', FMV Reports Series 10/1, Institute for Formal Models and Verification. Johannes Kepler University, (2010).

[4] A. Biere, 'Lingeling and friends at the SAT competition 2011', Technical Report FMV 11/1, Institute for Formal Models and Verification, Johannes Kepler University, (2011).

[5] N. Eén and N. Sörensson, 'An extensible SAT-solver', in *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pp. 502–518. Springer-Verlag, (2004).

[6] M. Gagliolo and J. Schmidhuber, 'Learning dynamic algorithm portfolios', *Annals of Mathematics and Artificial Intelligence*, **47**(3-4), pp. 295–328, (2006).

[7] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, 'Conflict-driven answer set solving', in *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 386–392. AAAI Press/The MIT Press, (2007).

[8] C. Gomes and B. Selman, 'Algorithm portfolios', *Artificial Intelligence*, **126**(1-2), pp. 43–62, (2001).

[9] L. Guo, Y. Hamadi, S. Jabbour, and L. Sais, 'Diversification and intensification in parallel SAT solving', in *Proceedings of the Sixteenth International Conference on Principles and Practice of Constraint Programming (CP'10)*, volume 6308 of *Lecture Notes in Computer Science*, pp. 252–265. Springer-Verlag, (2010).

[10] Y. Hamadi, S. Jabbour, and L. Sais, 'Control-based clause sharing in parallel SAT solving', in *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 499–504. AAAI Press/The MIT Press, (2009).

[11] Y. Hamadi, S. Jabbour, and L. Sais, 'ManySAT: a parallel SAT solver', *Journal on Satisfiability, Boolean Modeling and Computation*, **6**, pp. 245–262, (2009).

[12] M. Heule, M. Dufour, J. van Zwieten, and H. van Maaren, 'March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver', in *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, volume 3542 of *Lecture Notes in Computer Science*, pp. 345–359. Springer-Verlag, (2004).

[13] H. Hoos, 'Programming by optimisation', *Communications of the ACM*, **55**, pp. 70–80, (2012).

[14] H. Hoos, R. Kaminski, T. Schaub, and M. Schneider, 'aspeed: ASP-based solver scheduling', in *Technical Communications of the Twenty-eight*

International Conference on Logic Programming (ICLP'12). Leibniz International Proceedings in Informatics (LIPIcs), (2012). To appear.

[15] B. Huberman, R. Lukose, and T. Hogg, 'An economic approach to hard computational problems', *Science*, **275**, pp. 51–54, (1997).

[16] F. Hutter, H. Hoos, and K. Leyton-Brown, 'Automated configuration of mixed integer programming solvers', in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*, volume 6140 of *Lecture Notes in Computer Science*, pp. 186–202. Springer, (2010).

[17] F. Hutter, H. Hoos, and K. Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer-Verlag, (2011).

[18] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle, 'ParamILS: An automatic algorithm configuration framework', *Journal of Artificial Intelligence Research*, **36**, pp. 267–306, (2009).

[19] F. Hutter, H. Hoos, and T. Stützle, 'Automatic algorithm configuration based on local search', in *Proceedings of the Twenty-second National Conference on Artificial Intelligence (AAAI'07)*, pp. 1152–1157, AAAI Press, (2007).

[20] F. Hutter, H. Hoos, and K. Leyton-Brown, 'Parallel algorithm configuration', in *Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12)*, *Lecture Notes in Computer Science*. Springer-Verlag, (2012). To appear.

[21] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, 'ISAC – instance-specific algorithm configuration', in *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10)*, pp. 751–756. IOS Press, (2010).

[22] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan, 'Using case-based reasoning in an algorithm portfolio for constraint solving', in *Proceedings of the Nineteenth Irish Conference on Artificial Intelligence and Cognitive Science (AICS'08)*, (2008).

[23] M. Petrik and S. Zilberstein, 'Learning static parallel portfolios of algorithms', in *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM 2006)*, (2006).

[24] O. Roussel. Description of ppfolio, 2011. Available at `http://www.cril.univ-artois.fr/~roussel/ppfolio/solver1.pdf`, Last visited on 07-19-2012.

[25] M. Schneider and H. Hoos, 'Quantifying homogeneity of instance sets for algorithm configuration', in *Proceedings of the Sixth International Conference Learning and Intelligent Optimization (LION'12)*, *Lecture Notes in Computer Science*. Springer-Verlag, (2012). To appear.

[26] M. Soos, K. Nohl, and C. Castelluccia, 'Extending SAT solvers to cryptographic problems', in *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *Lecture Notes in Computer Science*, pp. 244–257. Springer-Verlag, (2009).

[27] W. Wei and C. Li. Switching between two adaptive noise mechanism in local search for SAT, 2009. Available at `http://home.mis.u-picardie.fr/~cli/EnglishPage.html`, Last visited on 07-19-2012.

[28] L. Xu, H. Hoos, and K. Leyton-Brown, 'Hydra: Automatically configuring algorithms for portfolio-based selection', in *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, pp. 210–216. AAAI Press, (2010).

[29] X. Yun and S. Epstein, 'Learning algorithm portfolios for parallel execution', in *Proceedings of the Sixth International Conference Learning and Intelligent Optimization (LION'12)*, *Lecture Notes in Computer Science*. Springer-Verlag, (2012). To appear.

An Efficiently Learnable Constructive Method for Graphs

Fabrizio Costa[1]

Introduction

Graph data structures allow us to model complex entities in a natural and expressive way. In the literature, several types of discriminative systems that can deal with graphs in input are known (e.g. recursive neural networks, graph kernels, etc), however, there are few generative or constructive approaches that can output graphs belonging with high probability to a desired distribution or class.

We argue that such systems are of great interest, and that novel and key problems in the field of Machine Learning (ML), and more generally in Artificial Intelligence (AI), can be addressed once such methods become viable. Let us digress a while and put things in perspective. Currently, the vast majority of models developed in the ML/AI research field are classifiers[2] (or regressors). These are used by practitioners of various disciplines (chemists, biologists, etc) mainly as proxies to replace expensive experimental measurements. In other words, ML tools are used to approximate hard to measure features (i.e. the biological activity of a molecule) on the basis of easy to measure features (i.e. the atomic composition and structure of a molecule). In the early days of AI and ML, the focus was on how to maximally exploit the information present in the handful of available data-points (mainly using domain knowledge to inject a strong bias both in the search strategy and in the hypothesis space). Nowadays however, the data bottleneck is disappearing in several domains (e.g. chemistry, biology, medicine). The dreaded consequence, as noted by [3], is that when sufficient data is available to cover the manifold, simple interpolation methods, such as the k-nearest neighbor technique, exhibit more than adequate predictive performance, rendering all other more sophisticated methods irrelevant. Further progress and increased ingenuity in experimental approaches are likely to exacerbate the issue. Nowadays, large scale and high-throughput experimental techniques can address directly many questions whose answer could previously only be approximated by computational methods (e.g. large screening to assess the biological activity of thousands of molecules, the assessment of large protein-protein interaction networks). In fact, it seems inevitable that in many fields the experimental approach will supersede the computational modeling strategy, and that it will do so at an accelerating rate[3].

[1] Bioinformatics Group, Department of Computer Science, University of Freiburg, Georges-Köhler-Allee 106, D-79110 Freiburg, Germany. Email: costa@informatik.uni-freiburg.de

[2] There are naturally many other types of contributions which do not fall under the classification type: feature selection, clustering and data mining, structured output prediction, to name a relevant few.

[3] To understand why, consider that the cost of several high-throughput techniques is decreasing at a faster rate than the cost for electronics predicted by Moore's laws.

The Challenge

One way for ML and AI to contribute to the advancement of other scientific disciplines, is to seek a type of task that cannot be easily solved by enhanced experimental techniques alone: one such task is the *learnable design problem* (LDP)[4]. A design (or generative) problem is formulated as the task to output instances belonging (with high probability) to a desired class. A learnable design or generative problem requires in addition that the class be (efficiently) modeled or learned starting from a finite sample of representatives instances.

Disregarding, initially, any limits derived from encoding and representational issues, we can list a range of interesting LDP application cases.

Given a set of game types ranked by a single player preference, one could automatically design a series of ever novel personalized games.

Given a set of molecules that have a desired biological activity one can design novel molecules that exhibit the same activity.

Given a set of bacteria whose metabolism byproduct is of utility (e.g. fermentation, carbon dioxide fixation, etc) one could design novel metabolic networks that have improved or multiple properties of interest.

In order to formulate these types of learnable design problems we need to address three main issues: 1) define a rich and expressive enough formalism to encode complex entities; 2) gain access to a sufficiently large representative set of design cases (and possibly also to a set of counter-examples); and finally 3) develop techniques that can efficiently learn the design principles and subsequently generate candidate design solutions.

We contend that we now have all the ingredients to start tackling the LDP. An adequate candidate for point 1) is the hyper-graph representation formalism, with which to encode arbitrary discrete entities and their relations. Point 2) is increasingly less critical given the current explosion of data (even of structured and relational type) available in machine readable formats. Point 3) is the subject of this work and is detailed in the following.

Related Work

The design problem is present, under different names and with important differences, in several research areas.

Operations research is often concerned with the identification of problem solutions (which can be at times regarded as design solu-

[4] We note that there exist however exceptions, i.e. a design problems that can be tackled in a pure experimental way; one such case is the design of *small molecules* when addressed by a *combinatorial chemistry* approach. In this case, the experimental approach materializes in hardware a typical software optimization algorithm.

tions) that satisfy some optimality condition (i.e. the cheapest circuit layout, the minimal length travel path). Traditionally however it assumes both the the objective function as well as the constraints to be given in an explicit form rather. (i.e. it does not include a learning stage)

Genetic algorithms and other derivative-free optimization techniques are also used to solve certain design problems (e.g. in mechanical and civil engineering fields), but they usually encode instances with fixed length vector representations and are thus not immediately suited for dealing with complex data.

The approaches that most explicitly address the LDP are grammar induction techniques. However, methods for learning expressive grammars are still not well developed, even in the case of simple string languages, let alone more complex domains like graphs or hyper-graphs. Graph grammars have been actively studied since the late 1960's, but few papers have dealt with their stochastic versions (needed to guarantee robustness in the presence of noise and outliers) and even fewer have dealt with the task of learning the grammar from a finite sample[5].

A few graph grammar induction methods, though, have been implemented. Usually they are not robust w.r.t. outliers or other form of structural noise [6], they tend to have high computational costs (i.e. they do not scale well to tens of thousands of graphs), they assume the structure of the grammar is given [9], or they are limited to context-free type of grammars, which possess nice properties of decidability, but suffer from limited expressive power [8].

The Quality Assessment Issue

A crucial issue, in the learnable design problem, is the model quality assessment. While in classification problems it is easy to devise metrics to compare the predicted class to the available *true* class, in the case of newly designed instances the issue becomes more complex. First of all, we have to deal with the lack of true class assignments. Resorting to an *oracle* can be difficult or at times impossible[6].

To tackle this issue we propose a simple yet effective strategy: given a set of examples and counter-examples, we induce the generative model, we train a binary classifier only on newly designed candidates, and we compute the predictive performance on a test set of known instances; the result is compared to the performance obtained by the same type of discriminative learner when trained on the original set. If the newly constructed instances belong to the same concept class, then both models should[7] perform equally well on the same test set. The size of the deviation can then be used as a measure of similarity between the true concept and the learned one.

Method

Here we propose a supervised constructive approach that, differently from current graph grammar induction techniques, is context-sensitive, is robust to outliers and is computationally efficient, with linear time complexity in both the model induction and the candidate generation phase.

The key notion is derived from that of *substitutability* [4]. Recently Clark and Eyraud [1] showed that certain types of grammars can be identified in the limit from positive data alone using the congruence classes of the language. The idea is to assume that all substrings that *always* occur in the same context (i.e. that are *congruent*) belong to the same implicit category and can therefore be substituted in the generative phase. We extend this notion in two ways: 1) we upgrade it from strings to graphs; and 2) we allow a local notion of context. More specifically we restrict the substitutable graphs to *neighborhood subgraphs* of given radius R, i.e. graphs induced by a root vertex v and all vertices that are within distance R from v. We then define the notion of *interface* as the difference between two co-rooted neighborhood subgraphs of different radii. The interface constitutes the local context for the inner co-rooted neighborhood graph that we call *core*. We call interface *thickness* T half of the difference of the two radii.

The model induction phase consists in the enumeration of all possible cores and their correspondent interfaces, rooted in all vertices of all positive instances. We then train a robust and fast graph kernel SVM discriminative model (introduced by Costa *et al.* in [2]) on a dataset containing both positive and negative examples. Finally, the construction of candidate graphs is achieved as an iterated substitution of congruent cores (i.e. cores with matching interface). The resulting graphs are evaluated in their entirety by the SVM; a simple beam search strategy is applied to control the size of the generated set. Initial findings support the idea that no sophisticated technique is required in order to escape local minima, as, given the size of the substituted subgraphs, a very large search space is in fact explored at each step. Finally, we show how the quantity $(R - T)/(R + T)$ can be regarded as the generative procedure's *creative tendency*, since a small radius and a large thickness result in very conservative substitutions supported by large contexts, while large radius and small thickness result in non-constrained substitutions.

We present encouraging experimental results in the chemoinformatics domain. Here the design task is the *de novo* construction of molecular graphs [5] that exhibit toxic properties [7]. The initial findings show a significant increase in predictive performance when we add 4K generated molecules to the initial dataset of comparable size, corresponding to a 14% relative error reduction for the area under the precision recall curve. Finally, we report how the generative procedure re-creates up to 5% of the original test set molecules.

[5] The main area of research has been rather the study of the generated language properties or the graph recognition problem given a specific type of grammar.

[6] E.g. assessing the activity of a newly proposed molecular graph requires the extremely complex and expensive step of molecular synthesis.

[7] Note that this is a sufficient albeit not necessary condition.

REFERENCES

[1] A Clark and R Eyraud, 'Polynomial identification in the limit of substitutable context-free languages', *Journal of Machine Learning Research*, (2007).

[2] F Costa and K De Grave, 'Fast neighborhood subgraph pairwise distance kernel', *Proceedings of the 26th International Conference on Machine Learning*, 255–262, (2010).

[3] Alon Halevy, Peter Norvig, and Fernando Pereira, 'The Unreasonable Effectiveness of Data', *IEEE Intelligent Systems*, **24**(2), 8–12, (2009).

[4] Zellig S. Harris, 'Distributional structure.', *Word*, (1954).

[5] M Hartenfeller and et al, 'De novo drug design', *Methods Mol Biol*, (2011).

[6] E Jeltsch and HJ Kreowski, 'Grammatical inference based on hyperedge replacement: a summary', *IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives*, 7, (1993).

[7] Jeroen Kazius, Ross McGuire, and Roberta Bursi, 'Derivation and validation of toxicophores for mutagenicity prediction', *Journal of Medicinal Chemistry*, **48**(1), 312–320, (2005).

[8] Jacek P. Kukluk, Lawrence B. Holder, and Diane J. Cook, 'Inference of edge replacement graph grammars', *International Journal on Artificial Intelligence Tools*, **17**(3), 539–554, (2008).

[9] T Oates, S Doshi, and F Huang, 'Estimating maximum likelihood parameters for stochastic context-free graph grammars', *Inductive Logic Programming*, 281–298, (2003).

Analyzing manuscript traditions using constraint-based data mining

Tara Andrews[1] and **Hendrik Blockeel**[2] and **Bart Bogaerts**[2] and **Maurice Bruynooghe**[2] and **Marc Denecker**[2] and **Stef De Pooter**[2] and **Caroline Macé**[1] and **Jan Ramon**[2]

Abstract. Data mining tasks and algorithms are often categorized as belonging to one of a few specific types: clustering, association rule discovery, probabilistic modeling, etc. For some time now, it has been recognized that concrete tasks do not always fit nicely in this categorization. The concepts of constraint-based data mining and inductive querying have been proposed to alleviate this problem; they offer more flexibility with respect to specifying the task. In this paper, we illustrate an approach that goes one step further: we show how a general-purpose declarative modeling language can be used to specify and solve data mining tasks in the area of philology. These tasks have the following properties: they are easily described in words; they are of real interest to philologists; they cannot be performed using standard querying or data mining systems; manually programming a solution for them is challenging, time-consuming and error-prone. We show that a prototype declarative programming framework, IDP, allows for easy modeling and efficient solving of these tasks. We conclude from this case study that the declarative modeling approach to data mining has a large potential and deserves further investigation.

1 Introduction

In data mining, many standard types of techniques exist, such as association rule discovery, decision tree induction, probabilistic modeling, clustering, etc. Once the data have been preprocessed into a format ready for analysis, these standard techniques can be run by simply "pushing the button", possibly after setting a few parameters. However, these techniques do not always produce exactly the type of results that the user wants. The term "constraint-based data mining" is often used to refer to data mining approaches where the user can impose constraints on the patterns he is looking for. For instance, association rule discovery algorithms may return only rules that fulfill certain syntactical constraints; clustering algorithms may accept must-link and cannot-link constraints that impose certain conditions on the returned clustering, etc. By allowing the user to specify such constraints, the data mining techniques become much more flexible: the user can tune them towards his own interests.

Instead of simply restricting the patterns returned by a standard data mining system, constraints may more generally be used to express background knowledge about the domain, or define the task one wants to solve. Nijssen and Guns [9], for instance, have shown that the classical task of frequent pattern discovery, possibly with constraints imposed on these patterns, can be defined completely as a constraint solving problem, and solved efficiently using a general-purpose solver. This shows that redefining classical data mining tasks in terms of constraints, and using a standard solver, can add a lot of versatility to the data mining process (users can define more precisely what they want), at a low efficiency cost.

Taking this one step further, one may try to find a single declarative language in which practically any data mining task could be defined, and solved efficiently, removing the need for many specialized systems. Even data mining tasks that do not belong to one of the predefined categories can then be solved quite easily. This approach is very similar to what is sometimes called "inductive querying" or "query-based data mining", but is more flexible in the sense that some data mining task descriptions may be too complicated to easily fit into a query, and more suitable for the more modular type of description that programming languages offer.

In this paper, we describe an approach where a declarative modeling language, $\mathrm{FO}(\cdot)^{\mathtt{IDP}}$, is used to describe the data, the background knowledge, and the data mining task. The application of interest is situated in the area of stemmatology, a sub-field of philology in which the history of manuscript traditions is studied. The mining tasks are motivated by real questions from philologists. The framework we use is called IDP. It provides seamless integration of $\mathrm{FO}(\cdot)^{\mathtt{IDP}}$ with a procedural language that can be used for reading files, preprocessing data, formatting the output, and calling the solver.

The remainder of this paper is structured as follows. Section 2 describes the application domain, stemmatology, in some more detail. Section 3 describes the IDP framework. In Section 4 we discuss several data mining tasks that are relevant for stemmatologists, and show how they can be solved using IDP. Section 5 presents conclusions.

2 Stemmatology

Before the invention of the printing press, texts were copied manually by scribes. This copying process was not perfect: scribes often modified texts, accidentally or intentionally. As a result, for many old texts the surviving copies vary significantly. No text written before the invention of the printing press, and even up to the end of the 18th century, when the habit of circulating texts in manuscript form practically dis-

[1] KU Leuven, Faculty of Arts
[2] KU Leuven, Department of Computer Science

appeared, can be read without a preliminary critical analysis of its material witnesses. This is the purpose of stemmatology. The Oxford English Dictionary defines the field as "the branch of study concerned with analysing the relationship of surviving variant versions of a text to each other, especially so as to reconstruct a lost original."

A stemma is a kind of "family tree" of a set of manuscripts. It indicates which manuscripts have been copied from which other manuscripts ("parents"), and which manuscript is the original source. It may include both extant (currently existing and available) and non-extant ("lost" or conjectured) manuscripts. Although stemmata are often assumed to be tree-shaped, they need not be. Sometimes, a manuscript has been copied partially from one manuscript, and partially from another, so it has multiple parents. In general, a stemma is a connected directed acyclic graph (DAG) [1].

The 19th century philologist Karl Lachmann was among the first to apply a principled method for reconstructing stemmata from sets of manuscripts [11]. Nowadays, a variety of methods exist. Many are borrowed from biology, where a similar problem, reconstruction of phylogenetic trees, is well-studied. However, these methods do not always fit the stemmatological context well. First, they assume that phylogenies are tree-shaped, while stemmata are DAGs.[3] Second, these trees contain only bifurcations, while stemmata can have multifurcations. Third, in most methods the trees are such that each extant copy is at a leaf of the tree, whereas in stemmatology one extant copy may be an ancestor of another (and hence should be an internal node). Fourth, stemmatologists often have additional information, for instance about the time or place of origin of a manuscript, which ideally should be taken into account. Research on new algorithms, better suited for the stemmatological context, continues [2].

Apart from reconstructing stemmata from data, stemmatologists are also interested in other types of analyses, which may, for instance, use a known stemma or a manually-constructed best-guess stemma as an input. These types of analysis have received even less attention, and they can be very diverse. The data mining tasks we address in this paper belong to this category. Multiple tasks will be addressed, but before discussing them in more detail, we first introduce the IDP framework.

3 IDP: an FO$(\cdot)^{\texttt{IDP}}$ knowledge-based programming environment

IDP [5] is a logic-based programming environment that extends the Lua [7] scripting language with support for constructing, manipulating, and performing inference with logical objects such as vocabularies, theories (in the FO$(\cdot)^{\texttt{IDP}}$ language), structures, and terms.

3.1 FO$(\cdot)^{\texttt{IDP}}$

The term FO$(\cdot)$ is used to denote the family of extensions of first-order logic (FO); for instance, FO(ID) is the extension of FO with inductive definitions. In this text, the focus lies on FO$(\cdot)^{\texttt{IDP}}$, the FO$(\cdot)$ language supported by the IDP framework. FO$(\cdot)^{\texttt{IDP}}$ extends FO with (among others) *types*, *arithmetic*, *aggregates*, *partial functions* and *inductive definitions*. This section focuses on what is needed for this paper; more information on FO$(\cdot)^{\texttt{IDP}}$ can be found in [13] and [4].

An FO$(\cdot)^{\texttt{IDP}}$ theory is a set of typed FO sentences and inductive definitions. We explain its syntax using a theory for solving a shortest path problem, presented in Figure 1. First, one declares a vocabulary, `sp_voc` in the example. It contains declarations of types (e.g., `node`), typed constants (e.g., `from`) and predicates (e.g., `edge(node,node)`). Next, one defines a theory over some vocabulary. It contains (well-typed) FO sentences constructed from symbols of the vocabulary and connectors and quantifiers, including `&`($\wedge$), `|`($\vee$), `~`($\neg$), `!`($\forall$), `?`($\exists$) and `~=`($\neq$). Finally, it also contains inductive definitions, sets of rules of the form `!` $x_1 \ldots x_n$ `: (P(`$t_1,\ldots,t_n$`) <- B)` where `B` is an FO formula. The semantics of inductive definitions is the well-founded semantics [12], as this semantics formalizes the intended meaning of all common forms of definitions [6].

The theory in Figure 1 expresses that the predicate `edgeOnPath` is the set of edges of a path from the node `from` to the node `to`. The following conditions must hold for such paths. (1) `edgeOnPath` is a subset of `edge`. (2) `from` and `to` are the first, respectively last node and hence, have no entering, respectively exiting edge in the path. (3) Each node on the path has at most one entering and at most one exiting edge in the path. Here, `?<2 y : edgeOnPath(y,x)` means that there are strictly less than 2 y's that have an edge to x in the path. This can be expressed also using an aggregate `#{ y : edgeOnPath(y,x)} < 2` or using a sentence `! y1 y2 : edgeOnPath(y1,x) & edgeOnPath(y2,x) => y1=y2`. (4) `from` can reach `to` using edges of the path and (5) `from` can reach each node on the path (e.g., `{ (from,to), (c,c)}` is not a path). The predicate `reaches(x,y)` is defined inductively.

The model semantics of FO and FO$(\cdot)^{\texttt{IDP}}$ is based on the notion of *structures*. A structure is an assignment of values to symbols: sets to type symbols, domain elements to constants, relations to predicate symbols and functions to function symbols. A *model* of a theory is an assignment that satisfies all expressions of the theory. In many problems, structures are useful to represent data. The structure defined in Figure 1 interprets the symbols `node`, `edge`, `from` and `to` from vocabulary `sp_voc`. Note that no value is specified for `edgeOnPath`, which means that this is a partial structure of `sp_voc`.

3.2 The IDP programming environment

Apart from vocabularies, theories and structures, the programming environment also contains *procedures* and *terms*. As said above, procedures are written in the language Lua and may call a range of predefined methods operating on the logical objects. The most relevant are the following:

- `sat(<theory>,<structure>)` is a boolean function that returns true iff the theory is satisfied by the structure.
- `modelexpand(<theory>,<structure>)` takes as input a theory over vocabulary Σ and a partial structure assigning values to some symbols in Σ, and returns a list of models of the theory that expand the partial structure.
- `minimize(<theory>,<structure>,<term>)` returns a list of models of the theory expanding the input structure in which the numerical term is minimal.

[3] Some methods return phylogenetic networks, but these represent uncertainty about the real tree, which is different from claiming that the network represents the actual phylogeny.

```
vocabulary sp_voc {
  type node
  from, to: node
  edge(node,node)
  edgeOnPath(node,node)
  reaches(node,node)
}
theory sp_theory: sp_voc {
  ! x y : edgeOnPath(x,y) => edge(x,y).
  ~(? x : edgeOnPath(x,from)) & ~(? x : edgeOnPath(to,x)).
  !x: (?<2 y: edgeOnPath(y,x)) & (?<2 y: edgeOnPath(x,y)).
  { reaches(x,y) <- edgeOnPath(x,y).
    reaches(x,y) <- reaches(x,z) & reaches(z,y). }
  reaches(from,to).
  ! x y : edgeOnPath(x,y) => reaches(from,y).
}
structure sp_struct: sp_voc {
  node = {A..D} // shorthand for A,B,C,D
  edge = {A,B; B,C; C,D; A,D}
  from = A
  to = D
}
term lengthOfPath: sp_voc {
  #{ x y : edgeOnPath(x,y) }
}
procedure main() {
  sols = minimize(sp_theory,sp_struct,lengthOfPath)
  if sols
  then print(sols[1])
  else print("No models exist.\n")
  end
}
```

Figure 1. `main()` finds the shortest path for the given data.

These methods are implemented with state-of-the-art technologies, such as the grounder GidL [14] and solver MiniSat(ID) [8]. The solver is an extended SAT solver with support for aggregate expressions, inductive definitions and branch-and-bound optimization. MiniSat(ID) also has support for finite domain constraints, using the propagation techniques described in [10] or, alternatively, interfacing with the Gecode Constraint Programming engine.

In Figure 1, `main()` performs the minimization inference on the path theory `sp_theory` and prints out the first model in the list if it exists. The term to be minimized here is called `lengthOfPath` and is declared as the number of pairs in `edgeOnPath`. A minimal model of this term is indeed a shortest path from A to D.

An IDP-program is a collection of declarations of vocabularies, theories, structures, terms and procedures. For an in-depth treatment of the framework, see [4].

3.3 Illustration: frequent itemset mining

Figure 2 shows how the task of frequent itemset mining can be described in IDP. We include it only as an example of how a classical data mining task can be defined in IDP; an in-depth study of the use of general-purpose solvers for frequent itemset mining is given by Nijssen and Guns [9].

The vocabulary declares two types `Transaction` and `Item`, the threshold `Freq`, the predicate `Includes(t,i)` which expresses that transaction `t` includes item `i`, and finally, the unary predicate `FrequentItemset`. The theory simply expresses that the number of transactions that include all items

```
vocabulary FrequentItemsetMiningVoc {
  type Transaction
  type Item
  Freq: int
  Includes(Transaction,Item)
  FrequentItemset(Item)
}
theory FrequentItemsetMiningTh: FrequentItemsetMiningVoc {
  #{t: !i: FrequentItemset(i) => Includes(t,i) } >= Freq.
}
structure Input : FrequentItemsetMiningVoc {
  Freq = 7 // threshold for frequent itemsets
  Transaction = { t1; ... ; tn } // n transactions
  Item = {i1 ; ... ;  im }       // m items
  Includes = {t1,i2; t1,i7; ...} // items of transactions
}
```

Figure 2. An IDP description of frequent itemset mining.

in `FrequentItemset` is at least `Freq`. The structure describes the data: it specifies the threshold, all transactions and items, and the `Includes` relation.

In any structure of this vocabulary that extends `Input` and satisfies the theory, `FrequentItemset` represents a frequent itemset. As a consequence, the task of computing all frequent itemsets is solved by letting IDP generate all such models.

4 Data mining tasks

4.1 Formalization of the context

A *tradition* is a set of manuscripts that are related in a particular way (specifically, they can be considered variants of the same text, the variation having been introduced through imperfect manual copying). A dataset represents one tradition. Each manuscript is described by a fixed set of attributes $A_1, \ldots, A_n$, each of which has a nominal domain $Dom(A_i)$. Typically, one attribute represents a particular location in the text, and the different elements of its domain represent the different variant readings at that location.

A *stemma* is a connected DAG $G(V, E)$, where V contains one element for each manuscript, and $(v, w) \in E$ if and only if manuscript w was (wholly or partially) copied from manuscript v.

Given an attribute A_i, we can label nodes in G with their value for A_i (observed or predicted). The whole labeling is then a partial or complete function $\lambda : V \to Dom(A_i)$. A labeling λ *extends* another labeling λ' if and only if, whenever $\lambda'(v)$ is defined, $\lambda(v)$ is also defined and $\lambda(v) = \lambda'(v)$.

4.2 The datasets

We will evaluate the feasibility of our declarative modeling approach on five datasets (`http://byzantini.st/stemmaweb/`). Three are artificial traditions, constructed with the purpose of testing stemmatological methods; for these, the correct stemma is known. They may be found at `http://www.cs.helsinki.fi/u/ttonteri/casc/data.html`. The other two (Sermon 158 and Florilegium) are real traditions, with stemmata that have been constructed according to current philological best practice. Table 1 gives an overview of the number of manuscripts and attributes for each tradition, as well as the number of edges in the DAG. Note that a tree with n nodes always has $n-1$ edges. Thus, three of the five stemmata are tree-shaped; two are "almost" tree-shaped, in the sense that they have few additional edges.

Table 1. The five traditions used in this work.

Name	nodes	edges	attributes
Notre Besoin	13	13	44
Parzival	21	20	122
Florilegium	22	21	547
Sermon 158	34	33	270
Heinrichi	48	51	1042

4.3 Task 1: Consistency checking

New variants are introduced when a scribe, intentionally or accidentally, changes a text. Often, a variant reading at one particular location is complicated enough to consider it unlikely that exactly the same reading has been introduced multiple times independently. We therefore introduce the following terminology. A manuscript is a *source* for an attribute if none of its parents have the same value for that attribute. An attribute is *consistent* with a stemma if the values observed for the attribute can be explained using only one source per value. Formally, given a stemma $G(V, E)$, an attribute A, and a (partial) labeling λ indicating which nodes in V have which value for A, A is consistent with G if and only if a complete labeling exists that extends λ and has one source for each value. Figure 3 illustrates these concepts.

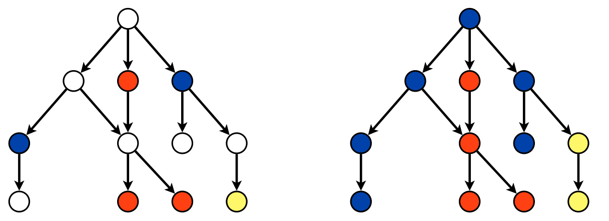


Figure 3. Left: a partial labeling showing for a given attribute which manuscripts have which value (indicated by colors). Right: a complete extension of that labeling with one source per label. Because such an extension exists, the attribute is consistent with the stemma.

Now consider the task of checking whether a given attribute is consistent with a given DAG. This requires searching for a complete extension of the given labeling that has one source per label. This problem is NP-hard [3] and does not reduce to any problem solved by standard data mining techniques.

The problem is easily modeled in IDP, however. Figure 4 shows an IDP program that determines for each attribute in each dataset whether it is consistent with the stemma. The problem of checking consistency of a single attribute with a stemma is defined under the header "Knowledge base". This definition is very simple: besides introducing the vocabulary (there are manuscripts; there are variant readings; manuscripts may be copied by other manuscripts; with

```
procedure main() {
  process("besoin")
  process("parzival")
  process("florilegium")
  process("sermon158")
  process("heinrichi")
}

/* ---------- Knowledge base ------------------------- */
vocabulary V {
  type Manuscript
  type Variant
  CopiedBy(Manuscript,Manuscript)
  VariantIn(Manuscript): Variant
}
vocabulary Vsrc {
  extern vocabulary V
  SourceOf(Variant): Manuscript
}
theory Tsrc : Vsrc {
  ! x : (x ~= SourceOf(VariantIn(x))) =>
        ? y: CopiedBy(y,x) & VariantIn(y) = VariantIn(x).
}

/* --------- Check whether sample fits stemma -------- */
procedure check(sample) {
  idpintern.setvocabulary(sample,Vsrc)
  return sat(Tsrc,sample)
}

/* ---------- Procedures for processing -------------- */
procedure process(name) {
  io.write("Processing ",name,".\n")
  local path = "data/"
  local stemmafilename = path..name..".dot"
  local samplefilename = path..name..".json"
  processFiles(stemmafilename,samplefilename)
}
procedure processFiles(stemmafilename,samplefilename) {
  local stemma,nbnodes,nbedges = readStemma(stemmafilename)
  io.write("Stemma has ",nbnodes," nodes and ",nbedges, " edges.\n")
  local nbp,nbs,time = processSamples(stemma,samplefilename)
  io.write("Found ",nbp," positive out of ",nbs," groupings ")
  io.write("in ",time," sec.\n")
}
procedure readStemma(stemmafilename) {
  ... // 19 lines
}
procedure processSamples(stemma,samplefilename) {
  ... // 23 lines
}
```

Figure 4. The IDP code for checking the consistency of all the attributes in a dataset with a hypothesized stemma for that dataset. The `.dot` and `.json` files contain the stemma and attribute-value-table, respectively.

each manuscripts is associated a variant; each variant has one source, which is a manuscript), it only states that if a manuscript is not the source of a variant, it must have a parent with that same variant.

Besides this declarative specification, the IDP program contains procedural code that loads the data files, builds for each dataset and attribute a structure that represents a partially labeled DAG, calls a solver to check the satisfiability of the theory for this structure (`sat(Tsrc, sample)`), and produces readable output. We include some of the procedural code to illustrate the seamless integration of declarative and procedural knowledge in IDP.

The IDP program determines consistency for all attributes and datasets in a matter of seconds:

```
> main()
Processing besoin.
Stemma has 13 nodes and 13 edges.
Found 26 positive out of 44 groupings in 0 sec.
Processing parzival.
Stemma has 21 nodes and 20 edges.
Found 45 positive out of 122 groupings in 1 sec.
Processing florilegium.
Stemma has 22 nodes and 21 edges.
Found 431 positive out of 547 groupings in 5 sec.
Processing sermon158.
Stemma has 34 nodes and 33 edges.
Found 64 positive out of 270 groupings in 4 sec.
Processing heinrichi.
Stemma has 48 nodes and 51 edges.
Found 1 positive out of 1042 groupings in 28 sec.
>
```

It is interesting to compare these results with earlier results obtained using a procedural implementation of the consistency check by one of the authors. This procedural implementation contained 370 lines of Perl code, using a graph library as working horse, and could not be shown correct. This was our main motivation for trying a declarative approach. Our declarative specification is more easily verified, is solved faster than with the Perl version, and eventually allowed us to show that the original procedural implementation was not correct [3]. This demonstrates the usefulness of the IDP framework for non-traditional types of data analysis.

4.4 Task 2: Determining the minimal number of independent sources

A remarkable result of the previous analysis was that for some of the artificial traditions, very few attributes were consistent with the stemma (for Heinrichi, 1 out of 1042). This indicates that either the artificial traditions are not representative for real traditions, or the assumption that each variant originates only once is not realistic.

Given that inconsistent attributes occur so often, one may wonder how often multiple introductions of the same variant must have occurred. In other words: what is the smallest number of sources needed to explain the observations?

This question is again easily expressed in IDP, now as a minimization problem. Figure 5 shows this. In the vocabulary, the function `SourceOf` (which allows only one source per label) is replaced by a predicate `IsSource`, which indicates whether a node x is a source or not. The theory simply defines `IsSource` as such; nothing else is needed. Further, a term `NbOfSources` is introduced that counts the number of sources, and a procedure is introduced that returns a model in which the number of sources is minimal. The procedure `minimize` performs model minimization, as explained before (in this case, it finds a complete and consistent labeling with a minimal number of sources). Apart from changing a call of `check(sample)` into `minSources(sample)`, and some output formatting, no procedural code needs to be changed.

Figure 6 shows part of the output for the Notre Besoin dataset. All datasets were processed in a few seconds, except for Heinrichi, which took about 5 minutes. Adding more constraints may further reduce processing time, but this was not investigated here.

```
/* ---------- Knowledge base ----------------------- */
vocabulary V {
  type Manuscript
  type Variant
  CopiedBy(Manuscript,Manuscript)
  VariantIn(Manuscript): Variant
}
vocabulary Vms {
  extern vocabulary V
  IsSource(Manuscript)
}
theory Tms : Vms {
  {!x: IsSource(x) <- ~?y: CopiedBy(y,x) &
                           VariantIn(y)=VariantIn(x).}
}
term NbOfSources : Vms {
  #{x:IsSource(x)}
}

/* --- Find model with minimal number of sources --- */

procedure minSources(sample) {
  idpintern.setvocabulary(sample,Vms)
  return minimize(Tms, sample, NbOfSources)[1]
}
```

Figure 5. IDP code for minimizing the number of sources required to explain the data. Function `SourceOf` is replaced by predicate `IsSource`; the term `NbOfSources`, which counts the number of sources, is introduced; and a procedure is introduced that returns a model with minimal `NbOfSources`.

```
Processing besoin.
Stemma has 13 nodes and 13 edges.
IsSource = { T2; U }
IsSource = { C; T2 }
IsSource = { D; J; L; M; T2; U; V }
... (40 output lines omitted)
IsSource = { B; F; J; T2 }
Minimized for 44 groupings in 0 sec.
```

Figure 6. IDP output indicating a minimal set of sources for each attribute in *Notre Besoin.*

4.5 Task 3: Sources versus reversions

Up till now, we said a manuscript introduces a variant if none of its parents have that variant. However, stemmatologists distinguish the case where a manuscript reverts to an older variant (which did not occur in the parents but occurs somewhere among the ancestors) from the case where it introduces a completely new variant. It may be interesting to minimize a cost function where sources have a higher cost than reversions.

This is again easily modeled in IDP. An inductive definition is provided for the predicate `IndirectAncestor`. Further, there are now three types of nodes: sources, reversions, and copies. This is modeled by introducing a function `ClassOf` that classifies nodes as one of `Source`, `Copy` or `Revert`. The theory and term-to-be-optimized are shown in Figure 7. In this case, some more changes to the procedural code (not shown) are required, for technical reasons beyond the scope of this paper. A part of the output for Notre Besoin is shown in Figure 8. Finding the model with minimal cost takes significantly longer than for the previous tasks: seconds to minutes for the first four datasets, and about 18 hours for Heinrichi. We currently do not know why it takes so much longer for

```
vocabulary Vcls {
  extern vocabulary V
  type Cost isa nat
  type Class
  Copy: Class
  Revert: Class
  Source: Class
  ClassOf(Manuscript): Class
  IndirectAncestor(Manuscript,Manuscript)
}
theory Tcls : Vcls {
  !x: (ClassOf(x)=Copy) <=>
      ?y: CopiedBy(y,x) & VariantIn(y) = VariantIn(x).
  !x: (ClassOf(x)=Revert) <=>
      ClassOf(x) ~= Copy &
      ?y: IndirectAncestor(y,x) &
          VariantIn(y) = VariantIn(x).
  {!x y: IndirectAncestor(x,y) <-
         ?z: CopiedBy(x,z) & IndirectAncestor(z,y).
   !x y: IndirectAncestor(x,y) <-
         ?z: CopiedBy(x,z) & CopiedBy(z,y).}
  NbOfSources = #{x: ClassOf(x)=Source}.
  NbOfReverts = #{x: ClassOf(x)=Revert}.
}
term TotalCost : Vcls {
  3 * NbOfSources + NbOfReverts
}
```

Figure 7. IDP code for minimizing the cost of a labeling, where each source has a cost of 3 and each reversion a cost of 1. The function `ClassOf` indicates which of three classes a node belong to: `Source`, `Revert` or `Copy`. The theory inductively defines the `IndirectAncestor` predicate, and defines the conditions under which a node has class `Copy` or `Revert`.

```
Processing besoin.
Stemma has 13 nodes and 13 edges.
ClassOf = {T2->s; U->s}
ClassOf = {C->s; T2->s}
ClassOf = {A->s; D->r; J->r; L->s; M->r; T2->s; U->r; V->r}
... (40 output lines omitted)
ClassOf = {A->s; B->s; F->r; J->r; T2->s}
Minimized for 44 groupings in 3 sec.
```

Figure 8. IDP output (edited for conciseness) showing sources (s) and reversions (r) for minimal-cost models in *Notre Besoin.*

Heinrichi. It may be possible to reduce runtime by adding more constraints to better guide the solver. Alternatively, approximate methods for optimization could be explored.

Tasks 2 and 3 demonstrate the ease with which new data mining tasks can be defined and solved, once the procedural code for preprocessing etc. is in place.

5 Conclusions

While many data mining tasks can be solved using off-the-shelf tools, some tasks deviate significantly from the standard ones and cannot be performed using any of the standard methods or query languages. Inductive query languages may provide a solution when the data mining task can be formulated as a relatively simple query. Here, we have explored an alternative approach that consists of defining the task using a declarative modeling language, then performing inference using advanced, built-in, constraint solving and optimization techniques. More concretely, the IDP framework has been used for addressing data mining tasks in stemmatology. As it turns out, IDP has the power and versatility to define these data mining tasks with relative ease, and solve them efficiently and provably correctly. Important elements of IDP that contribute to this are the ability to formulate constraints in full first-order logic, to include inductive definitions, to define aggregate functions, and to solve satisfiability and optimization problems. One opportunity for improvement is the minimization procedure, which might benefit from approximate methods. We conclude that declarative modeling frameworks such as IDP have a large potential for data mining, and this type of approaches deserves further investigation.

Acknowledgements

Research supported by Research Foundation - Flanders (FWO-Vlaanderen), KU Leuven CREA/10/004, and ERC Starting Researcher Grant 240186.

REFERENCES

[1] T. Andrews and C. Macé, ‘Beyond the tree of texts: Building an empirical model of scribal variation through graph analysis of texts and stemmata’, *In preparation* (2012).

[2] P. Baret, C. Macé, P. Robinson, C. Peersman, R. Mazza, J. Noret, E. Wattel, Van Mulken M., Robinson P., A. Lantin, P. Canettieri, V. Loreto, H. Windram, M. Spencer, C. Howe, M. Albu, and A. Dress, ‘Testing methods on an artificially created textual tradition.’, in *The evolution of texts: Confronting stemmatological and genetical methods*, 255–283, Istituti editoriali e poligrafici internazionali, Pisa (2006).

[3] H. Blockeel, B. Bogaerts, M. Bruynooghe, B. De Cat, S. De Pooter, M. Denecker, A. Labarre, J. Ramon, and S. Verwer, ‘Modeling machine learning and data mining problems with FO(.)’, in *Proc. 28th ICLP*, Leibniz International Proceedings in Informatics (2012). To appear.

[4] B. Bogaerts, B. De Cat, S. De Pooter, and M. Denecker. The IDP framework reference manual. `http://dtai.cs.kuleuven.be/krr/software/idp3/documentation`.

[5] S. De Pooter, J. Wittocx, and M. Denecker, ‘A prototype of a knowledge-based programming environment’, in *International Conference on Applications of Declarative Programming and Knowledge Management* (2011).

[6] M. Denecker and E. Ternovska, ‘A logic of nonmonotone inductive definitions’, *ACM Transactions on Computational Logic (TOCL)*, **9**(2), Article 14 (2008).

[7] R. Ierusalimschy, L.H. de Figueiredo, and W. Celes, ‘Lua – an extensible extension language’, *Software: Practice and Experience*, **26**(6), 635–652 (1996).

[8] M. Mariën, J. Wittocx, M. Denecker, and M. Bruynooghe, ‘SAT(ID): Satisfiability of propositional logic extended with inductive definitions’, in *Proc. SAT 2008*, volume 4996 of *LNCS*, pp. 211–224. Springer (2008).

[9] S. Nijssen and T. Guns, ‘Integrating constraint programming and itemset mining’, in *ECML/PKDD (2)*, volume 6322 of *Lect. Notes in Comp. Sc.*, pp. 467–482. Springer (2010).

[10] C. Schulte and P.J. Stuckey, ‘Efficient constraint propagation engines’, *ACM Transactions on Programming Languages and Systems*, **31**(1) (2008).

[11] S. Timpanaro and G.W. Most (translator), *The Genesis of Lachmann’s Method*, University of Chicago Press, 2005.

[12] A. Van Gelder, K.A. Ross, and J.S. Schlipf, ‘The well-founded semantics for general logic programs’, *Journal of the ACM*, **38**(3), 620–650 (1991).

[13] J. Wittocx, M. Mariën, and M. Denecker, ‘The IDP system: a model expansion system for an extension of classical logic’, in *LaSh*, pp. 153–165 (2008).

[14] J. Wittocx, M. Mariën, and M. Denecker, ‘Grounding FO and FO(ID) with bounds’, *Journal of Artificial Intelligence Research*, **38**, 223–269 (2010).

Column generation for exact BN learning: Work in progress

James Cussens[1]

1 Introduction

In existing integer linear programming (ILP) approaches to learning the structure of a Bayesian network (BN) from data ([1, 4, 2]) a binary variable $I(W \to u)$ is created for each BN variable $u \in V$ and each candidate parent set W. $I(W \to u)$ takes the value 1 iff W is the parent set for u in the optimal DAG. The number of candidate parent sets for any given BN variable is artificially restricted to keep n, the number of $I(W \to u)$ variables, reasonable and then a local score $c'(u, W)$ is pre-computed for each of them. With this approach the BN learning problem can be cast as in (1).

$$\begin{array}{l}\text{Instantiate the } I(W \to u) \text{ to maximise:} \\ \sum_{u,W} c'(u,W) I(W \to u) \\ \text{subject to the } I(W \to u) \text{ representing a DAG.}\end{array} \tag{1}$$

In this paper it will be more convenient to convert this into a minimisation problem, so set $c(u, W) = -c'(u, W)$ and consider

$$\begin{array}{l}\text{Instantiate the } I(W \to u) \text{ to minimise:} \\ \sum_{u,W} c(u,W) I(W \to u) \\ \text{subject to the } I(W \to u) \text{ representing a DAG.}\end{array}$$

The goal is thus to find a BN with the lowest negative score. To ensure that each BN variable has exactly one parent set the following $|V|$ constraints are used:

$$\forall u \in V : \sum_{W} I(W \to u) = 1 \tag{2}$$

and to ensure that the resulting graphs are acyclic, *cluster constraints* are added in the course of solving as *cutting planes*.

$$\text{Where } C \subseteq V : \sum_{u \in C} \sum_{W : W \cap C = \emptyset} I(W \to u) \geq 1 \tag{3}$$

Cluster constraints were introduced in [4] and were later used in [2]. It is not necessary to add all (exponentially many) cluster constraints. A lower bound on the negative score of the optimal BN (the dual bound) increases (or occasionally remains constant) as cluster constraints are added. Once a BN has been found whose score equals this lower bound it follows that an optimal BN has been found.

2 The simplex algorithm

The inequalities (3) can be replaced by equations using 'slack' variables w_C:

$$-w_C + \sum_{u \in C} \sum_{W : W \cap C = \emptyset} I(W \to u) = 1 \tag{4}$$

where each w_C is constrained to be positive ($w_C \geq 0$). Let $\mathcal{C}$ be the current set of clusters. The full set of constraints for the ILP problem is then a collection of $|V| + |\mathcal{C}|$ linear equations involving $(n + |\mathcal{C}|)$ variables. Following the notation and presentation given in [5], let $\mathbf{x}$ be the vector of these ILP variables (in some arbitrary order), then these constraints can be written in matrix from as:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{5}$$

where $\mathbf{A}$ is a $(|V| + |\mathcal{C}|) \times (n + |\mathcal{C}|)$ matrix. Note from (2) and (4) that $\mathbf{b}$ will be a column of ones.

Consider a solution to (5) corresponding to a completely unconnected graph. The variables $I(\emptyset \to u)$ will have value 1, all other $I(W \to u)$ variables will have value 0 and the slack variables will have the values determined by the equations (4). Let $\zeta = \sum_{u,W} c(u, W) I(W \to u)$ be a variable representing the objective value and let $\bar{\zeta} = \sum_u c(u, \emptyset)$ be the constant which is the score for the completely unconnected graph. It is not difficult to see that we can write the objective function and the constraints as follows:

$$\begin{aligned}
\zeta &= \bar{\zeta} + \sum_{u,W : W \neq \emptyset} [c(u,W) - c(u,\emptyset)] I(W \to u) && (6)\\
I(\emptyset \to u) &= 1 - \sum_{u,W : W \neq \emptyset} I(W \to u) && (7)\\
w_C &= |C| - 1 - \sum_{u \in C} \sum_{W : W \cap C \neq \emptyset} I(W \to u) && (8)
\end{aligned}$$

where (7) represents an equation for each $u \in V$ and (8) represents an equation for each cluster $C \in \mathcal{C}$. Essentially (7) has been used to eliminate all $I(\emptyset \to u)$ variables from the RHS of each equation.

The equations (6–8) are called a *dictionary* [6]. The variables on the RHS of the equations, which are all set to 0, are the *non-basic* variables, those on the LHS of (7) and (8) are the *basic* variables. Note that there are $n - |V|$ non-basic variables and $|V| + |\mathcal{C}|$ basic variables.

To improve the value of ζ the simplex algorithm considers the coefficients $[c(u, W) - c(u, \emptyset)]$ of the non-basic variables on the RHS of (6). These coefficients are called the *reduced costs* of the non-basic variables. If a variable has negative reduced cost then raising its values from zero will improve (reduce) the value of ζ. So a variable with negative reduced cost is increased until one of the basic variables becomes set to zero. These variables are called the *entering* and *leaving* variables respectively. The entering variable moves from non-basic to basic ('enters the basis') and the leaving variable moves from basic to non-basic. A new dictionary is created (implicitly in practice) where the entering variable is removed from all RHS, being replaced by a linear expression involving the leaving variable. This ensures that all basic variables and the objective value are represented

[1] University of York, email:james.cussens@york.ac.uk

as linear functions of non-basic variables (which means that reduced costs are available for all non-basic variables). This process continues until no non-basic variable has negative reduced cost at which point the linear program is solved.

3 Column generation

The key idea behind column generation is that *it is not necessary to* explicitly *represent non-basic variables*. So an as-yet-nonexistent variable can be viewed as a non-basic variable which we have yet to consider as a possible entrant into the basis. Note that variables correspond to columns of the matrix $\mathbf{A}$ in (5) so that generating a new variable corresponds to generating a new column in that matrix. The goal of column generation is to create a new variable with negative reduced cost.

We consider now how to compute reduced costs. Recall that at each iteration of the simplex algorithm we have $|V| + |\mathcal{C}|$ basic and $n - |V|$ non-basic variables. Write $\mathbf{x} = (\mathbf{x_B}, \mathbf{x_D})$ where $\mathbf{x_B}$ are the basic variables and $\mathbf{x_D}$ the non-basic. Decompose the objective coefficient vector $\mathbf{c}$ similarly: $\mathbf{c} = (\mathbf{c_B}, \mathbf{c_D})$. Let $\mathbf{B}$ be the submatrix of the original $\mathbf{A}$ matrix formed by selecting the columns of $\mathbf{A}$ corresponding to $\mathbf{x_B}$. Let $\mathbf{D}$ be the corresponding submatrix for non-basic variables. Note that $\mathbf{B}$, called the *basis matrix*, is a $(|V| + |\mathcal{C}|) \times (|V| + |\mathcal{C}|)$ square matrix. We can now compute $\lambda^T = \mathbf{c_B^T B^{-1}}$. λ is the vector of dual values for each row (=linear constraint) in $\mathbf{A}$. The vector of reduced costs for non-basic variables, denoted $\mathbf{r_D}$, can then be computed using $\mathbf{r_D} = \mathbf{c_D} - \lambda^T \mathbf{D}$.

Note that the dual vector λ is determined by the current basis. So to compute the reduced cost of a potential new variable we just need its objective coefficient value and its coefficient for each original linear constraint (= row of $\mathbf{A}$).

We now construct an ILP to identify a new variable with negative reduced cost. A new variable $I(W \to u)$ is determined by a choice of the child u and also the parents W. Let $I_{\text{ch}}(u)$ indicate that u is chosen as the child and let $I_{\text{pa}}(u)$ represent that u is chosen as a parent. We have the obvious constraints:

$$\sum_{u \in V} I_{\text{ch}}(u) = 1 \quad (9)$$

$$\forall u \in V : I_{\text{ch}}(u) + I_{\text{pa}}(u) \leq 1 \quad (10)$$

A new variable $I(W \to u)$ will appear in a cluster constraint for $C \in \mathcal{C}$ (4) with coefficient 1 iff $u \in C$ and $v \notin C$ for each $v \in W$. Create a variable x_C for each $C \in \mathcal{C}$ indicating whether the new variable is involved in the constraint for C. We have:

$$x_C \geq \sum_{u \in C} I_{\text{ch}}(u) - \sum_{u \in C} I_{\text{pa}}(u) \quad (11)$$

$$\sum_{u \in C} I_{\text{ch}}(u) \geq x_C \quad (12)$$

$$1 - \sum_{u \in C} I_{\text{pa}}(u) \geq x_C \quad (13)$$

Let λ_C be the dual value corresponding to the constraint for cluster C. Let λ_u be the dual value for the convexity constraint (2) for variable u. The reduced cost for a new variable $I(W \to u)$ is then:

$$c(u, W) - \sum_u \lambda_u I_{\text{ch}}(u) - \sum_{C \in \mathcal{C}} \lambda_C x_C \quad (14)$$

To find the best new variable to introduce we want to minimise (14) subject to constraints (9-13): an ILP. However, in (14), $c(u, W)$ the objective coefficient for the new variable, is unknown. The sensible option is to view $c(u, W)$ as an additional (real-valued) ILP variable. For the column generation method to produce useful new variables, it is essential that we can put a reasonably tight lower bound on $c(u, W)$ in terms of the variables $I_{\text{ch}}(u)$, $I_{\text{pa}}(u)$ and x_C and a small number of constants easily computable from the data. Since we only have a bound on $c(u, W)$ solving the ILP will generate a new variable together with an over-optimistic value for its reduced cost. If even this over-optimistic value is positive it follows that there are no new variables worth introducing and so the current set of $c(u, W)$ variables are all we need to identify the optimal BN. If the over-optimistic value is negative there is at least the possibility that the *actual* reduced cost is also negative. This makes it worth the effort to consult the data to compute the true reduced cost. If this turns out to be negative, the variable is created. Otherwise constraints should be added to rule out this choice for $c(u, W)$ and the ILP re-solved in the hope of finding a different variable with a (true) negative reduced cost.

The key to this approach is the tightness of the sought bound. Recent work in a companion paper [3] has produced the following lower bound on $c(u, W)$:

$$\begin{aligned} & -c(u, W) \\ \leq \quad & (\alpha - qr/2) \log r + (q(r-1)/2) \log(\frac{\alpha}{q}) \\ & -NH_{\tilde{p}}(u|W) \quad (15) \\ & -\frac{1}{2} \sum_k \log\left(n_k + \frac{\alpha}{qr}\right) + \frac{1}{2} \sum_j \log\left(n_j + \frac{\alpha}{q}\right) \quad (16) \end{aligned}$$

where: α is the *effective sample size*, r is the arity of u, q is the product of the arities of the variables in W, $H_{\tilde{p}}(u|W)$ is the conditional entropy of u given W according to a certain distribution $\tilde{p}$, N is the size of the data, the n_j are the counts in the contingency table for the variables W and the n_k the contingency table counts for $\{u\} \cup W$. A useful bound for the data-dependent (15) is available, but not, at present, for (16). It is hoped that with further work this lower bound on $c(u, W)$ can be used to allow effective column generation.

REFERENCES

[1] James Cussens. Maximum likelihood pedigree reconstruction using integer programming. In *Proceedings of the Workshop on Constraint Based Methods for Bioinformatics (WCB-10)*, Edinburgh, July 2010.

[2] James Cussens. Bayesian network learning with cutting planes. In Fabio G. Cozman and Avi Pfeffer, editors, *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pages 153–160, Barcelona, 2011. AUAI Press.

[3] James Cussens. An upper bound for BDeu local scores. Submitted, May 2012.

[4] Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 358–365, 2010. Journal of Machine Learning Research Workshop and Conference Proceedings.

[5] David G. Luenberger and Yinyun Ye. *Linear and nonlinear programming*. Springer, 2008.

[6] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, third edition, 2008.

Constraint-based Learning for Text Categorization

Savatore Frandina and **Claudio Saccà** and **Michelangelo Diligenti** and **Marco Gori** [1]

Abstract. Text categorization automatically assigns a document to its underlying topics. Documents are typically represented as bag-of-words, and machine learning based approaches have been shown to provide effective and scalable solutions by learning from examples. However, a limiting factor in the application of these approaches relies on the large number of examples required to train a classifier working on large taxonomies of classes. This paper presents a method to integrate prior knowledge that is typically available on the learning task into a text classifier based on kernel machines. The presented solution deals with any prior knowledge represented as first-order logic (FOL) and, thanks to the generality of this formulation, can be used to express relations among the input patterns, known semantic relationships among the output categories and input-output rules. The kernel machine mathematical apparatus is re-used to cast the learning problem into a primal optimization of a function composed of the loss on the supervised examples, the regularization term, and a penalty term deriving from converting the knowledge into a set of continuous constraints. The experimental results, performed over the popular CORA dataset, show that the proposed approach over-performs both SVMs and state-of-the-art semi-supervised techniques in multi-label text classification problem.

1 Introduction

Text categorization decides the topics of a document based on its representation. Documents are typically represented as bag-of-words, and classical machine learning tools can be used to perform the classification after having trained a model from examples. SVMs, a special class of kernel machines, have been proved to be one of the most versatile machine learning approaches to text categorization, providing near state-of-the-art accuracy on many datasets while requiring little tuning. This paper presents a novel method to perform text categorization using any prior knowledge available. The approach is based on a framework that integrates kernel machines and logic to solve multi-task learning problems. The kernel machine mathematical apparatus allows casting the learning problem into a primal optimization of a function composed of the loss on the supervised examples, the regularization term, and a penalty term deriving from forcing the constraints converting the logoc. This naturally allows to get advantage of unsupervised patterns in the learning task, as the degree of satisfaction of the constraints can be measured on unsupervised data. This paper assumes that prior knowledge is available both in terms of know relations among the input patterns (as it happens for Web documents connected via hyperlinks or scientific papers connected via citations), known semantic relationships among the output categories (for example to model an ontology) and in terms of input-output rules. We assume that this knowledge can be represented in first-order logic (FOL). The connections between logic and machine learning have been the subject of many investigations, like [1] which studies the relationships between symbolic and sub-symbolic models in AI. A broader coverage of the field with emphasis on the connections with inductive logic programming is in [2]. A related approach to combining first-order logic and probabilistic graphical models in a single representation are Markov Logic Networks [3]. In [4], the well-known inductive logic programming system FOIL is combined with kernel methods by leveraging FOIL search for a set of relevant clauses. This model, called kFOIL, can be used to solve either classification or regression tasks. However, one main limitation of the reviewed approaches is the lack of tight integration between the machine learning which deals with the perceptual representation of the patterns and the prior knowledge on the patterns and classes. The only direct attempt [5] is limited to rules on the perceptual space (input-output). The idea of centering the theory around the general and unified notion of constraints turns out to be a very straightforward way of bridging logic and kernels, since it is possible to express most classic logic formalisms by constraints and supervised examples as used in most learning machines are just a special instance of (soft) constraints. The experimental results, performed over the popular CORA dataset, confirm that the proposed approach performs better than supervised (SVMs) and semi-supervised (Laplacian SVM, Transductive SVM) approaches in a multi-label classification problem.

The paper is organized as follows: the next section introduces learning from constraints with kernel machines. The translation of any FOL knowledge into real-valued constraints is described in section 3, and some experimental results are reported in section 4. Finally some conclusions and are drawn.

2 Learning with constraints

We consider a multitask learning problem in which the input is a tuple $\mathcal{X} = \{x_j | x_j \in \mathcal{D}_j, j = 1, \ldots, n\}$, being $\mathcal{D}_j$ the domain of the values for the j-th attribute. The learning task considers a set of functions $\{\tau_k(x_{j(1,k)}, \ldots, x_{j(n_k,k)}) | k = 1, \ldots, T,\ x_{j(l,k)} \in \mathcal{X},\ \tau_k \in \mathcal{T}_k\}$ taking a subset of the data as input. Some of the functions may be known a priori whereas others must be inferred from examples. In general, we assume that the attributes in each domain are described by a real valued vector of features that are relevant to solve the tasks at hand. Hence, it holds that $\mathcal{D}_j = I\!R^{d_j}$ and $\tau_k : I\!R^{d_{j(1,k)}} \times \ldots \times I\!R^{d_{j(n_k,k)}} \rightarrow I\!R$. For the sake of compactness, in the following we will indicate by $\boldsymbol{x}_k = [x'_{j(1,k)} \cdots x'_{j(n_k,k)}]' \in I\!R^{d_k}$, where $d_k = \sum_{l=1,\ldots,n_k} d_{j(l,k)}$, the input vector for the k-th task.

We consider the case when the tasks functions τ_k have to meet a set of constraints that can be expressed by the functionals $\phi_h : \mathcal{T}_1 \times \ldots \times \mathcal{T}_T \rightarrow [0, +\infty)$ such that $\phi_h(\tau_1, \ldots, \tau_T) = 0 \quad h = 1, \ldots, H$ must hold for any valid choice of $\tau_k \in \mathcal{T}_k, k = 1, \ldots, T$.

[1] Dipartimento di Ingegneria dell'Informazione, Università di Siena, Italy, email: {claudio.sacca,frandina,michi,marco}@dii.unisi.it

In order to define the learning task, we suppose that each task function τ_k can be approximated by a f_k in an appropriate Reproducing Kernel Hilbert Space $\mathcal{H}_k$. Therefore, the learning procedure can be cast as an optimization problem that aim at computing the optimal functions $f_1 \in \mathcal{H}_1, \ldots, f_T \in \mathcal{H}_T$, where $f_k : I\!R^{d_{j(1,k)}} \times \ldots \times I\!R^{d_{j(n_k,k)}} \to I\!R$, $k = 1, \ldots, T$. In the following, we will indicate by $\boldsymbol{f} = [f_1, \ldots, f_T]'$ the vector collecting these functions.

We consider the classical learning formulation as a risk minimization problem. Assuming that the correlation among the input features $\boldsymbol{x}_k$ and the desired task function output y_k is modeled by a joint probability distribution $p_{(\boldsymbol{x}_k,y_k)}(\boldsymbol{x}_k, y_k)$, the risk associated to a hypothesis $\boldsymbol{f}$ is measured as,

$$R[\boldsymbol{f}] = \sum_{k=1}^{T} \lambda_k^\tau \cdot \int_{D_k} L_k^e\left(f_k(\boldsymbol{x}_k), y_k\right) p_{(\boldsymbol{x}_k,y_k)}(\boldsymbol{x}_k, y_k)\, d\boldsymbol{x}_k\, dy_k$$

where D_k is the domain of the x_k, $\lambda_k^\tau > 0$ is the weight of the risk for the k-th task and $L_k^e\left(f_k(\boldsymbol{x}_k), y_k\right)$ is a loss measuring the fitting quality of $f_k(\boldsymbol{x}_k)$ with respect to the target y_k. Common choices for the loss function are the quadratic function especially for regression tasks, and the hinge function for binary classification tasks.

The regularization term can be written as $N[\boldsymbol{f}] = \sum_{k=1}^{T} \lambda_k^r \cdot ||f_k||^2_{\mathcal{H}_k}$, where $\lambda_k^r > 0$ can be used to weight of the regularization contribution for the k-th task.

Clearly, if the tasks are uncorrelated, the optimization of the objective function $R[\boldsymbol{f}] + N[\boldsymbol{f}]$ with respect to the T functions $f_k \in \mathcal{H}_k$ is equivalent to T stand-alone optimization problems for each function. However, if we consider a problem for which some correlations among the tasks are known a priori and coded as rules, we can enforce also these constraints in the learning procedure. Following the classical penalty approach for constrained optimization, we can embed the constraints by adding a term that penalizes their violation. Since the functionals $\phi_h(\boldsymbol{f})$ are strictly positive when the related constraint is violated and zero otherwise, the overall degree of constraint violation of the current hypothesis $\boldsymbol{f}$ can be measured as

$$V[\boldsymbol{f}] = \sum_{h=1}^{H} \lambda_h^v \cdot \phi_h(\boldsymbol{f}) ,$$

where the parameters $\lambda_h^v > 0$ allow us to weight the contribution of each constraint. It should be noticed that, differently from the previous terms considered in the optimization objective, the penalty term involves all the functions and, thus, explicitly introduces a correlation among the tasks in the learning statement. Finally, we can add together all the contributions yielding the objective E[$\boldsymbol{f}$] = R[$\boldsymbol{f}$] + N[$\boldsymbol{f}$] + V[$\boldsymbol{f}$].

Since the distributions $p_{(\boldsymbol{x}_k,y_k)}(\boldsymbol{x}_k, y_k)$, $k = 1, \ldots, T$ needed to determine $R[\boldsymbol{f}]$ are usually not known, we apply the common assumption to approximate them through their empirical realizations. This requires to have a set of examples drawn from these unknown distributions. Basically, the learning set will contain a set of labeled examples for each task k: $\mathcal{L}_k = \left\{\left(\boldsymbol{x}_k^i, y_k^i\right) | i = 1, \ldots, \ell_k\right\}$. The unsupervised examples are collected in $\mathcal{U}_k = \{\boldsymbol{x}_k^i | i = 1, \ldots, u_k\}$, while $\mathcal{S}_k^L = \{\boldsymbol{x}_k | \left(\boldsymbol{x}_k, y_k\right) \in \mathcal{L}_k\}$ collects the sample points that are in the supervised set for the k-th task. The set of the supervised and unsupervised points for the k-th task is $\mathcal{S}_k = \mathcal{S}_k^L \bigcup \mathcal{U}_k$.

Given an input object we can assume that also a partial labeling can be provided, i.e. it is not required to specify the targets for all the considered tasks for each sample corresponding to the i-th instance $\mathcal{X}^i$ of the input tuple. In the following we will refer to the unsupervised set $\mathcal{U} = \{\mathcal{X}^i | \exists k \ : \ \boldsymbol{x}_k^i \in \mathcal{U}_k\}$.

In general, the functionals $\phi_h(\boldsymbol{f})$ implementing the constraints involve all the values computed by the functions in $\boldsymbol{f}$ on their whole domains making training difficult. Hence, as in the case of the risk, we assume that these functionals can be conveniently approximated by considering an appropriate sampling in the function domains. In particular, the exact constraint functional will be replaced by an approximation exploiting only the values of the unknown functions $\boldsymbol{f}$ computed for the points in $\mathcal{U}$: $\phi_h(\boldsymbol{f}) \approx \hat{\phi}_h(\mathcal{U}, \boldsymbol{f})$.

Thus, the given learning problem is cast in a semi-supervised framework where it is assumed that a set of (partially) labeled examples is exploited together with a lset of unlabeled examples. Given the available supervised examples in $\mathcal{L}_k$ and an unsupervised sample $\mathcal{U}_k$, $k = 1, \ldots, T$, the objective function considering the empirical risk and the empirical penalty is,

$$\begin{aligned} E_{emp}[\mathbf{f}] &= \sum_{k=1}^{T} \frac{\lambda_k^\tau}{|\mathcal{L}_k|} \sum_{\left(\boldsymbol{x}_k^j, y_k^j\right) \in \mathcal{L}_k} L_k^e\left(f_k(\boldsymbol{x}_k^j), y_k^j\right) + \\ &+ \sum_{k=1}^{T} \lambda_k^r \cdot ||f_k||^2_{\mathcal{H}_k} + \sum_{h=1}^{H} \lambda_h^v \cdot \hat{\phi}_h(\mathcal{U}, \boldsymbol{f}) . \end{aligned} \tag{1}$$

3 Translation of first-order logic clauses into real-valued constraints

We focus attention on knowledge-based descriptions given by first-order logic (FOL–KB). In the following, we indicate by $\mathcal{V} = \{v_1, \ldots, v_N\}$ the set of the variables used in the KB, with $v_s \in \mathcal{D}_s$. Given the set of predicates used in the KB: $\mathcal{P} = \{p_k | p_k : \mathcal{D}_{s(1,k)} \times \ldots \times \mathcal{D}_{s(n_k,k)} \to \{true, false\},\ k = 1, \ldots, T\}$, the clauses will be built from the set of atoms $\mathcal{A} = \{p_{k(i)}(v_{s(1,k(i))}, \ldots, v_{s(n_{k(i)},k(i))}) | i = 1, \ldots, m,\ p_{k(i)} \in \mathcal{P},\ v_{s(j,k(i))} \in \mathcal{V}\}$, where the i-th atom is an instance of the $k(i)$-th predicate for which the j-th argument is assigned to the variable $v_{s(j,k(i))} \in \mathcal{D}_{s(j,k(i))}$. In the following, for the sake of compactness, we will indicate by $\boldsymbol{v}_{a_i} = [v_{s(1,k(i))}, \ldots, v_{s(n_{k(i)},k(i))}]$ the argument list of the atom $a_i \in \mathcal{A}$.

With no loss of generality, we restrict our attention to FOL clauses in the PNF form, where all the quantifiers ($\forall, \exists$) and their associated quantified variables are placed at the beginning of the clause. The quantifier-free part of the expression is equivalent to an assertion in propositional logic for any given assignment of the quantified variables. Since any propositional expression can be written in *Conjuctive Normal Form* (CNF), we can assume that all FOL expressions are in PNF-CNF form,

$$\overbrace{[\forall\exists] v_{s(1)} \ldots [\forall\exists] v_{s(Q)}}^{\text{Quantified Portion}} \overbrace{\bigwedge_{c=1,\ldots,C} \left(\bigvee_{d=1,\ldots,D_c} [\neg]\, a_{i(c,d)}(\mathbf{v}_{a_{i(c,d)}}) \right)}^{\text{Quantifier-free CNF expression } E_0(\mathbf{v}_{E_0}, \mathcal{P})} ,$$

where $a_{i(c,d)} \in \mathcal{A}$ is an atom and the variables $v_{s(q)} \in \mathcal{V}$, $q = 1, \ldots, Q$ constitute the set of the quantified variables. The quantifier-free expression $E_0(\boldsymbol{v}_{E_0}, \mathcal{P})$ depends on the list of arguments $\boldsymbol{v}_{E_0} = [v_{s(1,E_0)}, \ldots, v_{s(n_{E_0},E_0)}]$ corresponding to the variables used in all the atoms $a_{i(c,d)}$, i.e. $v_{s(j,E_0)} \in \{v_q \in \mathcal{V} | \exists c, d\ v_q \in \text{args}(a_{i(c,d)})\}$ where args$(a_{i(c,d)})$ is the set of the variables $v_{a_{i(c,d)}}$ used as arguments in the atom $a_{i(c,d)}$.

We assume that the task functions f_k are exploited to implement the predicates in $\mathcal{P}$ and each variable in $\mathcal{V}$ maps to the attributes defining the tuple $\mathcal{X}$ on which the functions f_k are defined.

The FOL–KB will contain a set of clauses corresponding to expressions with no free variables (i.e. all the variables appearing in the expression are quantified) that are assumed to be *true* in the considered domain. These clauses can be converted into a set of constraints as in that can be enforced during the kernel based learning process. The conversion process of a clause into a constraint functional consists of the following three steps:

I. PREDICATE SUBSTITUTION: substitution of the predicates with their continuous implementation realized by the functions $\boldsymbol{f}$ composed with a squash function, mapping the output values into the interval $[0, 1]$ such that the value 0 is associated with *false* and 1 with *true*.. In particular, the atom $a_i(\boldsymbol{v}_{a_i})$ is mapped to $\sigma(f_{k(i)}(\boldsymbol{v}_{a_i}))$, where $\sigma : I\!\!R \to [0, 1]$ is a monotonically increasing squashing function. A natural choice for the squash function is the piecewise linear mapping $\sigma(y) = \min(1, \max(y, 0))$, this is indeed the function that was employed in the experimental results.

II. CONVERSION OF THE PROPOSITIONAL EXPRESSION: conversion of the quantifier-free expression where all atoms are grounded as detailed in subsection 3.1. In our context the propositional logic clause to be generalized into a continuous function is grounded with the output values of the functions applied on a pattern (if unary), or on a vector of patterns if n-ary.

III. QUANTIFIER CONVERSION: conversion of the universal and existential quantifiers as shown in section 3.2.

3.1 Logic expressions and their continuous representation

As studied in the context of fuzzy logic and symbolic AI, different methods can be used for the conversion of a propositional expression into a continuous function with $[0, 1]$ input variables.

T-norms. In the context of fuzzy logic, t-norms [6] are commonly used as a generalization of logic clauses to continuous variables. A t-norm is a function $t : [0, 1] \times [0, 1] \to I\!\!R$, that is commutative, associative, monotonic and featuring a neutral element 1 (i.e. $t(x, 1) = x$). A *t-norm fuzzy logic* is defined by its t-norm $t(x, y)$ that models the logic AND and a function modeling the negation of a formula. For example, the negation of x corresponds to $1 - x$ in the Lukasiewicz norm. Many different t-norm logics have been proposed in the literature like the *product t-norm* $t(x, y) = x \cdot y$ and the *minimum t-norm* defined as $t(x, y) = \min(x, y)$. Once defined the t-norm functions corresponding to the logical AND and NOT, these functions can be composed to convert any arbitrary logic proposition.

Mixture of Gaussians. A different approach based on mixtures of Gaussians has been proposed in [7] in the context of symbolic learning using neural networks. Unlike t-norms, this approach generalizes the logic clause without making any independence assumption among the variables. In particular, let us consider a propositional logic clause involving n logic variables. The logic clause is equivalent to its truth table containing 2^n rows, each one corresponding to a configuration of the variables. The continuous function approximating the clause is based on a set of Gaussian functions, each one centered on a configuration corresponding to the true value in the truth table. The mixture function sums all the Gaussians:

$$t(\mathbf{x}) = \sum_{[\mathbf{c_1},\ldots,\mathbf{c_n}] \in \mathcal{T}} \exp\left(-\frac{||[\mathbf{x_1},\ldots,\mathbf{x_n}]' - [\mathbf{c_1},\ldots,\mathbf{c_n}]'||^2}{2\sigma^2}\right)$$

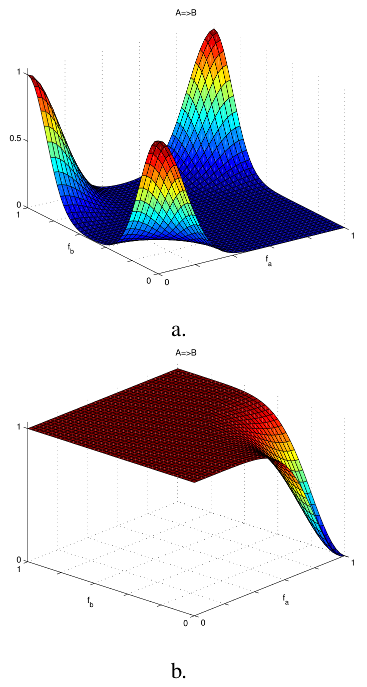


Figure 1: Function resulting from the conversion of $a \Rightarrow b$ using PGAUSS (a.) and NGAUSS (b.).

where $\mathbf{x} = [\mathbf{x_1}, \ldots, \mathbf{x_n}]$ is a vector containing the variables in the clause and $\mathcal{T}$ is the set of all possible configurations of the input variables corresponding to a true truth value. We indicate as $PGAUSS$ this conversion procedure. The value of $t(x, y)$ will decrease depending on the distance from the closest configuration verifying the clause. Each configuration verifying the constraint is always a global maximum of t when using a small enough σ value. See [7] for a complete discussion on how to select σ.

An alternative approach, that we indicate as NGAUSS, is to represent the false values of the truth table of the clause. In this case, one negative Gaussian is centered on each configuration of variables yielding a false value of the considered clause. A bias value equal to 1 is introduced to obtain a default true value when distant from a false configuration:

$$t(\mathbf{x}) = \mathbf{1} - \sum_{[\mathbf{c_1},\ldots,\mathbf{c_n}] \in \mathcal{F}} \exp\left(-\frac{||[\mathbf{x_1},\ldots,\mathbf{x_n}]' - [\mathbf{c_1},\ldots,\mathbf{c_n}]'||^2}{2\sigma^2}\right),$$

where $\mathcal{F}$ is the set of input configurations corresponding to a *false* value in the truth table.

Figure 1 shows the functions obtained by converting the clause $a \Rightarrow b$ using PGAUSS and NGAUSS. Any formula can be converted using both forms but, depending on the formula, one mixture can be more compact.

Hypercube. This class of conversion methods considers the n-dimensional space formed by associating each logic propositional variable in a clause to a dimension of the space. This builds a hypercube associating each configuration of the variables in the truth table to a vertex. Let $tt(\mathbf{c})$ be a function mapping a configuration $\mathbf{c} = [c_1, \ldots, c_n]$ to its truth value: $tt(\mathbf{c}) = 1$ if $c \in \mathcal{T}$ and $tt(\mathbf{c}) = 0$ if $c \in \mathcal{F}$. Let's now consider a continuous generalization of the logic variables in the $[0,1]$ range. The generalized truth value of a point $\mathbf{x}$ can be computed as weighted average of the values in the vertices: $t(\mathbf{x}) = \sum_{k=1}^{2^n} w_k(\mathbf{x}) \cdot tt(\mathbf{c}_k)$ where $\mathbf{c}_k$ is a tuple corresponding to k-th configuration in the truth table. Depending on the selection of the weights $w_k(\mathbf{x})$ different conversion schema are obtained. For example, the *Hypercube-Closest-Vertex* norm selects the truth value of the closest vertex as:

$$w_k(\mathbf{x}) = \begin{cases} D(\mathbf{x},\mathbf{c}_k) & D(\mathbf{x},\mathbf{c}_k) \leq D(\mathbf{x},\mathbf{c}_j) \;\; j=1,\ldots,2^N, j \neq k \\ 0 & otherwise \end{cases}$$

where $D(\mathbf{x},\mathbf{c}_k)$ is the Euclidean distance between $\mathbf{x}$ and vertex $\mathbf{c}_k$. The *Hypercube-Distance* norm weights vertices inversely proportional to the generalized Hamming distance from the input $\mathbf{x}$:

$$w_k(\mathbf{x}) = \prod_{i=1}^{n} |c_{ki} - c_i|$$

where c_{ki} is the i-th element of $\mathbf{c}_k$. The main advantage of this latter norm is that any point in a hyperplane merging two or more vertexes with the same truth value tt will also be assigned to tt. For example, when converting the rule $A \Rightarrow B \wedge C$, any of the 4 vertexes corresponding to $A = 0$ are associated to a true value in the truth table (e.g. $tt(0, c_1, c_2) = 1$). Therefore, $t(0, x_1, x_2) = 1$ for any $x_1 = [0,1], x_2 = [0,1]$ meaning that any point on the hypercube face will satisfy the constraint. This property allows to build constraints that are easier to satisfy as they do not introduce any unnecessary requirement.

Any of the above norms allow the mapping of any arbitrary quantifier-free expression $E(\boldsymbol{v}_E, \mathcal{P})$ to a functional constraint can be written $\varphi_E(\boldsymbol{v}_E, \boldsymbol{f}) = 0$, depending on all the variables collected in the argument list $\boldsymbol{v}_E = [v_{s(1,E)}, \ldots, v_{s(n_E,E)}]$ and on the predicates implemented by the functions $\boldsymbol{f}$.

3.2 Quantifier conversion

The quantified portion of the expression is processed recursively by moving backward from the inner quantifier in the PNF expansion.

Let us consider the universal quantifier first. The universal quantifier expresses the fact that the expression must hold for any realization of the quantified variable v_q. When considering the real–valued mapping of the original boolean expression, the universal quantifier can be naturally converted measuring the degree of non-satisfaction of the expression over the domain $\mathcal{D}_{j(q)}$ where the feature vector $x_{j(q)}$, corresponding to the variable v_q, ranges. This measure can be implemented by computing the overall distance of $\varphi_E(\boldsymbol{v}_E, \boldsymbol{f})$, that is the degree of violation associated to the quantified expression, from the constant function equal to 0 (this is the only value for which the constraint is always verified), over the domain $\mathcal{D}_{j(q)}$. Measuring the distance using the infinity norm yields

$$\forall v_q \; E(\mathbf{v}_E, \mathcal{P}) \to \|\varphi_E(\boldsymbol{v}_E, \boldsymbol{f})\|_\infty = \sup_{v_q \in \mathcal{D}_{j(q)}} |\varphi_E(\boldsymbol{v}_E, \boldsymbol{f})| \; , \quad (2)$$

where the resulting expression depends on all the variables in $\boldsymbol{v}_E$ except v_q. Hence, the result of the conversion applied to the expression $E_q(\boldsymbol{v}_{E_q}, \mathcal{P}) = \forall v_q \; E(\boldsymbol{v}_E, \mathcal{P})$ is a functional $\varphi_{E_q}(\boldsymbol{v}_{E_q}, \boldsymbol{f})$, assuming values in $[0,1]$ and depending on the set of variables $\boldsymbol{v}_{E_q} = [v_{s(1,E_q)}, \ldots, v_{s(n_{E_q},E_q)}]$, such that $n_{E_q} = n_E - 1$ and $v_{s(j,E_q)} \in \{v_r \in \mathcal{V} | \exists i \; v_r = v_{s(i,E)}, \; v_r \neq v_q\}$. The variables in $\boldsymbol{v}_{E_q}$ need to be quantified or assigned a specific value in order to obtain a constraint functional depending only on the functions $\boldsymbol{f}$.

Theorem 1. *Let $E(v, \mathcal{P})$ be an FOL expression with no quantifiers depending on the variable v. Let $t_E(v, \boldsymbol{f})$ be the continuous representation of E, where f_k corresponds to p_k, $k = 1, \ldots, T$. If $f_k \in \mathbb{C}^0$, $k = 1, \ldots, T$, then $\|1 - t_E(v, \boldsymbol{f})\|_p = 0$ iff $\forall v \; E(v, \mathcal{P})$ is* true.

Proof: See [8].

Theorem 1 shows that there is a duality between an universally quantified expression and its continuous generalization. If we consider the conversion of the PNF representing a FOL constraint without free variables, the variables are recursively quantified until the set of the free variables is empty. In the case of the universal quantifier we apply again the mapping described previously. The existential quantifier can be realized by starting from eq. (2), and enforcing the De Morgan law ($\exists v_q \; E(\boldsymbol{v}_E, \mathcal{P}) \iff \neg \forall v_q \; \neg E(\boldsymbol{v}_E, \mathcal{P})$) to hold also in the continuous mapped domain:

$$\exists v_q \; E(\boldsymbol{v}_E, \mathcal{P}) \quad \to \quad \inf_{v_q \in \mathcal{D}_{j(q)}} \varphi_E(\boldsymbol{v}_E, \boldsymbol{f})$$

It is generally complex to compute this expression, since the conversion of the quantifiers requires to ground the the quantified variables over the whole domain. We assume that the computation can be approximated by exploiting the empirical realizations of the feature vectors. Hence, the quantifiers exploiting the infinity norm are approximated using the empirical distribution $\mathcal{S}_{x_j}$ for x_j as:

$$\begin{aligned} \forall v_q \; E(\boldsymbol{v}_E, \mathcal{P}) \quad &\to \quad \max_{v_q \in \mathcal{S}_{x_{j(q)}}} |\varphi_E(\boldsymbol{v}_E, \boldsymbol{f})| \\ \exists v_q \; E(\boldsymbol{v}_E, \mathcal{P}) \quad &\to \quad \min_{v_q \in \mathcal{S}_{x_{j(q)}}} |\varphi_E(\boldsymbol{v}_E, \boldsymbol{f})| \; . \end{aligned}$$

In description logics, it is common to define an $\exists_n$ operator, which generalizes the existential operator to from one to n elements. While FOL can also indirectly express $\exists_n$, it may be convenient and more compact to provide a direct translation of the $\exists_n$ operator as:

$$\exists_n v_q \; E(\boldsymbol{v}_E, \mathcal{P}) \quad \to \quad \min(n)_{v_q \in \mathcal{S}_{x_{j(q)}}} |\varphi_E(\boldsymbol{v}_E, \boldsymbol{f})| \; ,$$

where $\min(n)$ is the n-th minimum value over the set. It is possible to use a different norm to convert the universal quantifier, for example using norm-1: $\forall v_q \; E(\boldsymbol{v}_E, \mathcal{P}) \quad \to \sum_{v_q \in \mathcal{S}_{x_{j(q)}}} |\varphi_E(\boldsymbol{v}_E, \boldsymbol{f})|$.

Please note that $\varphi_E([], \boldsymbol{f})$ will always reduce to a form that depends on the realizations of the functions over the data point samples. The solution to the optimization task defined by equation 1 with constraints evaluated over a finite set of data points can be computed by considering the following extension of the Representer Theorem [9], see [8] for details and a proof.

Theorem 2. *Given a multitask learning problem for which the task functions $f_1, \ldots, f_T$, $f_k : I\!R^{d_k} \to I\!R$, $k = 1, \ldots, T$, are assumed to belong to the Reproducing Kernel Hilbert Spaces $\mathcal{H}_1, \ldots, \mathcal{H}_T$, respectively, and the problem is formulated as $[f_1^*, \ldots, f_T^*] = argmin_{f_1 \in \mathcal{H}_1, \ldots, f_T \in \mathcal{H}_T} E_{emp}[f_1, \ldots, f_T]$ where $E_{emp}[f_1, \ldots, f_T]$ is defined as in equation (1), then each function f_k^* in the solution can be expressed as*

$$f_k^*(\boldsymbol{x}_k) = \sum_{\boldsymbol{x}_k^i \in \mathcal{S}_k} w_{k,i}^* K_k(\boldsymbol{x}_k^i, \boldsymbol{x}_k)$$

where $K_k(\boldsymbol{x}_k', \boldsymbol{x}_k)$ is the reproducing kernel associated to the space $\mathcal{H}_k$, and $\mathcal{S}_k$ is the set of the available samples for the k-th function.

Therefore, the optimal solution can be expressed as a kernel expansion over the data points. In fact, since the constraint is represented by $\varphi_E([], \boldsymbol{f}) = 0$ in the definition of the learning objective function, it is possible to substitute $\hat{\phi}(\mathcal{U}, \boldsymbol{f}) = \varphi_E([], \boldsymbol{f})$.

3.3 Special cases

Transductive SVMs [10] correspond to a special case of the proposed framework, where a logic clause imposes that any predicate should be either true or false. While this rule is always verified in standard logic, it is not verified in fuzzy logics. Therefore,

$$\forall x \;\; P(x) \vee \neg P(x)$$

forces function f_P estimating predicate P to assume values that are away from the separating surface even on unsupervised data. As typically done in Transductive SVMs, it is possible to avoid unbalanced solutions by imposing that

$$\exists_n x \;\; P(x) \wedge \exists_m \, x \;\; \neg P(x) \quad : \quad n + m = N$$

where N is the total number of patterns and n and m are estimated from the supervised examples: $n = N \cdot n^P_{pos}/(n^P_{pos} + n^P_{neg})$ where n^P_{pos} and n^P_{neg} are the number of positive and negative supervised examples for predicate P, respectively.

Manifold regularization [11] assumes that the classification functions should be regular over the manifold built over the input data distribution. Laplacian SVMs are a effective semi-supervised approach to train SVMs under the manifold regularization assumption. Let us introduce a predicate $R(x, y)$ which holds true if and only if x, y are connected on the manifold. R is typically a known predicate which is built using geometrical properties. The manifold assumption in a logic setting, where two connected points should either both true or false can be expressed as:

$$\forall x \;\; R(x, y) \Rightarrow (P(x) \wedge P(y)) \vee (\neg P(x) \wedge \neg P(y)) \; .$$

3.4 Stage-based learning

The optimization of the overall error function is performed in the primal space using gradient descent [12]. Unlike when only considering supervised examples, the objective function is non-convex due to the constraint term. In order to face the problems connected with the presence of sub-optimal solutions, the optimization problem was split in two stages. In a first phase, as commonly done by kernel machines it is performed regularized fitting of the supervised examples. Only in a second phase, the constraints are enforced since requiring a higher abstraction level. This solution has intriguing connections with results of developmental psychology, since it is well-known that many animals experiment stage-based learning [13]. From a pure optimization point of view, the first stage with the correspondent guarantee of convergence to an optimal solution makes possible to approach the global basin of attraction, while the second stage refines learning beginning from a good initialization. The different constraints can also be gradually introduced. As common practice in constraint satisfaction tasks, more restrictive constraints should be enforced earlier. As metric of how restrictive a logic constraint is, it is possible to use the portion of true configurations of the corresponding clause.

4 Experimental results

The experimental results have been carried out on a subset of the CORA dataset [2]. The CORA dataset is composed by a set of entities and their relations to allow experimenting with machine learning approaches which can cope with relations. Entities are authors and scientific papers, even if only papers are considered in our experimental setting. CORA assigns to each paper a set of categories (multi-label classification task), selected from a taxonomy of classes. The goal of our experiments is to predict the categories assigned to each paper.

For our experiments, we have created a dataset of scientific publications by selecting the 3 first-level (in the taxonomy) categories which have the highest number of papers. Then all the papers in the child classes of the selected higher level classes have been selected as well. All the papers not belonging to at least one of these categories have been discarded, from the remaining papers a random sample of 1000 papers has been performed. Each paper is then associated with a vectorial representation containing its title represented as bag-of-words. We assume that each category is associated to a predicate, taking a paper as input, which holds true if and only if the paper belongs to the category. We indicate as $C_i(\cdot)$ the predicate for the i-th class of the dataset.

Five folds have been generated by selecting $n\%$ of the papers for which supervisions are kept (n=10,20,30,40 over different experiments) as training set. 15% of the papers of the initial dataset have been been inserted into the validation set, while the remaining papers have been removed of the supervisions and used for testing.

The knowledge base collects different collateral information which is available on the dataset. For example, CORA makes available a list of citations for each papers, our algorithm can exploit these relations assuming that a citation represents a common intent between the papers that are therefore suggested to belong to the same set of categories. This can be expressed via a set of 10 clauses (one per category) such that foreach $i = 1, \ldots, 10$:

$$\forall x \in \mathcal{P} \; \forall y \in \mathcal{P} \;\; Cite(x, y) \Rightarrow (C_i(x) \wedge C_i(y)) \vee (\neg C_i(x) \wedge \neg C_i(y))$$

where $\mathcal{P}$ is the domain of all papers in the dataset and $Cite(x, y)$ is a binary predicate which holds true iff paper x cites paper y. This set of clauses will smooth the value of the estimated predicates over the citation graph. This effect is very similar to what would be done by manifold regularization or other similar techniques [14].

		10%	20%	30%	40%
SVM	Recall	0.473	0.521	0.576	0.624
	Precision	**0.714**	0.756	0.793	0.796
	F1	0.569	0.617	0.667	0.700
SBR	Recall	**0.672**	**0.692**	**0.741**	**0.770**
	Precision	0.673	0.741	0.773	0.804
	F1	**0.672**	**0.715**	**0.756**	0.787
TSVM	Recall	0.617	0.672	0.696	0.725
	Precision	0.602	0.677	0.695	0.711
	F1	0.608	0.674	0.695	0.718
LSVM	Recall	0.615	0.660	0.702	0.738
	Precision	0.669	0.744	0.770	0.814
	F1	0.641	0.699	0.734	0.774

Table 1: Precision, recall and F1 metrics averaged over 5 runs using SVM, Transductive SVM (TSVM), Laplacian SVM (LSVM) and Semantic Based Regularization (SBR) using only citation and taxonomic constraints varying the number of supervised patterns. Metrics in bold represent statistically significant gains (95%) over all the other classifiers.

Other clauses can be inserted to model the relationships among the classes. For example, the CORA taxonomy can be used to build clauses stating that if the predicate associated to a leaf node is true, then all the predicates associated to the nodes up to the root should be true as well:

$$\forall \, x \in \mathcal{P} \;\; C_i(x) \Rightarrow pa\,[C_i]\,(x)$$

where $pa\,[C_i]$ is the father category of C_i in the taxonomy. Furthermore, the following rule defines a close world assumption $\forall \; x \in$

[2] Download at http://people.cs.umass.edu/∼mccallum/data.html

$\mathcal{P}\ c_1(x) \vee c_2(x) \vee c_3(x)$, where c_1, c_2, c_3 are the three classes in the first-level of the taxonomy. Overall, a total number of 8 clauses was used to model the taxonomy. Other external semantic knowledge can be inserted as well using common knowledge about the environment. For example, let $HasWord$ be a given predicate holding true iff document x has word "neural", it is possible to associate the presence of the term to either the category Artificial Intelligence or Biology as:

$$\forall\, x \in \mathcal{P}\ \ HasWord(x, Neural) \Rightarrow C_{ai}(x) \vee C_{bio}(x)$$

where C_{ai}, C_{bio} indicate the predicate for Artificial Intelligence, and biology, respectively. 45 clauses of this kind have been added to the knowledge base.

The logic translation of the transductive rule as described in section 3.3 can also be added for each category (10 total). Therefore, the overall knowledge base is composed of 93 FOL clauses[3]. In our experimental setting, the Hypercube-distance norm has been use to convert all clauses, except the transductive rules for which the Hypercube-Nearest-Vertex norm has been employed. The norm-1 has been used to convert all clauses, except for citations rules for which the infinity norm was used. Stage-base learning was used to train all the SBR classifiers.

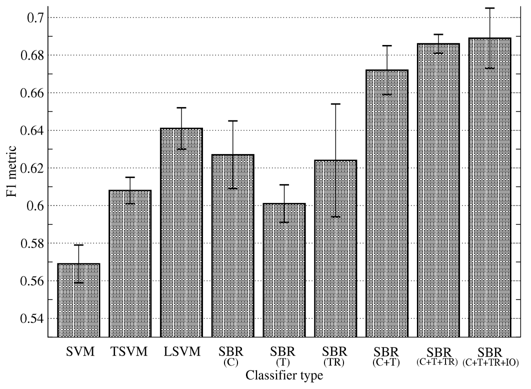


Figure 2: F1 (average over five runs) using SVM, TSVm , LSVM and SBR with different set of rules: taxonomic (T), citation (C), transductive (TR), input-output rules (IO) and different combinations of these over the dataset with 10% supervised.

For each subsample size of the training set, one classifier has been trained. As a comparison, we also trained for each set a standard SVM (using only the supervised labels), a Transductive SVM (implemented in the svmlight software package) and Laplacian SVM using the citations to build the manifold of data. When training with the knowledge base the stage-based procedure described in section 3.4 has been used. The validation set has been used to select the best values for λ^r and λ^c (same value for each function e.g. $\lambda^r = \lambda_i^r\ \ i = 1, \ldots, 10$ and $\lambda^c = \lambda_i^c\ \ i = 1, \ldots, 10$). The precision, recall and F1 results have been compute as an average over five different samples of the dataset. Table 1 summarizes the results for a different number of supervised data when using only citations and taxonomic logic clauses. SBR provides a statistically significant F1 gain for two datasets over four and the highest average F1 score for the other datasets.

Figure 2 shows a detailed breakdown of the effect of the single rules on the 10%-supervised dataset as an average over five runs. Citation, transductive, taxonomy and input-output constraints all contribute to improve over standard SVMs. However, F1 is maximized when all constraints are added at the same time. All the SBR classifiers combining multiple clauses provide gains over SVM, TSVM and LSVM that are statistically significant (95%).

5 Conclusions and future work

This paper presents a text categorization approach, which is able to provide a tight integration of prior knowledge and the classical kernel machine apparatus. The approach is very general, as any knowledge expressed in FOL can be considered, including relational knowledge among patterns and classes, semantic knowledge like ontologies and input-output rules. The experimental results have been carried out on the CORA dataset and they show the effectiveness of the proposed approach on a multi-label classification problem.

Acknowledgments. This research was partially supported by the research grant PRIN2009 "Learning Techniques in Relational Domains and Their Applications" (2009LNP494) from the Italian MURST.

REFERENCES

[1] P. Hitzler, S. Holldobler, and A. K. Sedab, "Logic programs and connectionist networks," *Journal of Applied Logic*, vol. 2, no. 3, pp. 245–272, 2004.
[2] L. D. Raedt, P. Frasconi, K. Kersting, and S. M. (Eds), *Probabilistic Inductive Logic Programming*, vol. 4911. Springer, Lecture Notes in Artificial Intelligence, 2008.
[3] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, no. 1–2, pp. 107–136, 2006.
[4] N. Landwehr, A. Passerini, L. D. Raedt, and P. Frasconi, "kfoil: Learning simple relational kernels," in *Proceeding of the AAAI-2006*, 2006.
[5] G. Fung, O. Mangasarian, and J. Shavlik, "Knowledgebased support vector machine classifiers," in *Proceedings of Sixteenth Conference on Neural Information Processing Systems (NIPS)*, (Vancouver, Canada), 2002.
[6] E. Klement, R. Mesiar, and E. Pap, *Triangular Norms*. Kluwer Academic Publisher, 2000.
[7] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, no. 1, pp. 5–32, 1996.
[8] M. Diligenti, M. Gori, M. Maggini, and L. Rigutini, "Bridging logic and kernel machines," *Machine Learning*, pp. 1–32, 2011.
[9] B. Scholkopf and A. J. Smola, *Learning with Kernels*. Cambridge, MA, USA: MIT Press, 2001.
[10] V. N. Vapnik, *Statistical learning theory*. Wiley, NY, 1998.
[11] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *The Journal of Machine Learning Research*, vol. 7, p. 2434, 2006.
[12] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, no. 5, pp. 1155–1178, 2007.
[13] J. Piaget, *La psychologie de l'intelligence*. Armand Colin, Paris, 1961.
[14] S. Peters, L. Denoyer, and P. Gallinari, "Iterative Annotation of Multi-relational Social Networks," in *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining*, pp. 96–103, IEEE, 2010.

[3] The vectorial representation of the patterns (both in SVMlight and our format) and the list of rules in a format suitable for our software simulator can be downloaded (together with the full software bundle) from https://sites.google.com/site/semanticbasedregularization/home

Instance-specific Parameter Tuning via Constraint-based Clustering

Lindawati and **Hoong Chuin LAU** and **Feida ZHU** [1]

Abstract. The performance of heuristic algorithm, is highly dependent on its parameter configuration. To automatically obtain good parameter configurations, we propose SufTra, a novel approach for instance-specific parameter tuning that utilizes the *Suf*fix tree data structure to represent *Tra*jectories of a Local Search algorithm. SufTra extracts compact features from search trajectories using Suffix Tree and filters these features using user specified-constraints. Then, it clusters the problem instances and computes the parameter configurations. Given an arbitrary testing instance, we obtain a suitable parameter configuration by mapping it to these clusters. Experimental evaluations of our approach on the Quadratic Assignment Problem (QAP) show that our approach offers significant improvement over existing parameter tuning algorithms. We also analyse the cluster quality, demonstrating an almost perfect match between our cluster results with the existing natural classifications.

1 Introduction

Good parameter configurations are critically important to ensure heuristic algorithms to be efficient and effective. Existing approaches for *automated parameter tuning* (also called *automated algorithm configuration* or *automated parameter optimization*) fall into two categories: *model-free* and *model-based*. Some *model-free* approaches can handle a large number of numerical and even categorical parameters (for example GGA [2], F-Race [4] and ParamILS [9]). *Model-based* approaches, on the other hand, offers statistical insights into the correlation of parameters with regard to algorithm performance. A recent example of the model-based approach is SMAC [8].

The challenge with parameter tuning is that different problem instances require different parameter configurations [10, 12, 19]. One way to enhance model-based approaches is to incorporate *instance features* to produce instance-specific parameter configurations. Unfortunately, finding *instance features* itself is often tedious and domain-specific, requiring a re-examination of features for each new problem. The research problem is to discover a **generic instance-specific automated parameter tuning** scheme that can perform as well as those exploiting problem-specific features.

In this work, we propose an approach to address this problem based on data mining and machine learning concepts. We focus our attention on tuning local search algorithms. In essence, we perform clustering of training instance's search trajectories, which is defined as a path of solutions discovered by the target algorithm as it searches through its neighborhood search space [7], and perform tuning on the clusters. For this purpose, we make use of a powerful data structure, namely suffix tree [5]. The nice characteristic of our work is that we can obtain these trajectories from the target local search algorithm with minimal additional computation effort.

Motivated by previous work [11] where human constraints give significant improvement to its solution, we also involve user-specified constraints to guide the clustering process. Although in some well-known problems such as QAP, user can specify the constraints very easily, for other problems, user-specified constraints are often hard to enumerate. Hence, we design our approach to anticipate partial (rather small) or no user-specified constraints. Note that user-specified constraints are different from instance features.

It is interesting to note that our approach does not make use of an explicit formulation (such as linear or Gaussian regression) that maps instances to clusters, which may be very hard if not impossible to derive. Instead, we exploit the *rich instance-specific* search trajectories as a proxy for the fitness landscape which is correlated with algorithm performance [15]. Instances are clustered based on these generic features using predictive modeling. This form of clustering preserves *rich features* that represent the individual instances within it.

Our approach improves the work of [12] that captures similarity using a single (and relatively short) segment through out the entire sequence, and works only on short and small number of sequences due to its inherent computational bottleneck. In contrast, our approach is capable of retrieving similarity across multiple segments with linear-time complexity. Using a Suffix Tree data structure and user-specified constraints, our approach can efficiently and effectively form better and tighter clusters and hence improve the overall performance of the underlying target algorithm.

We conduct experiments on the Quadratic Assignment Problem (QAP). The experiments show that our approach offers encouraging results for both cluster quality and overall performance compared against existing approaches. For the overall runtime, our approach is significantly (more than ten times) faster compared to the approach in [12] when the search trajectories are long and the number of training instances is large.

2 Preliminaries

To avoid confusion, we refer the algorithm whose performance is being tuned/configured as the *target algorithm* and the one that is used to tune/configure it as the *configurator*. We measure the target algorithm performance based on the quality of their solutions. We define function $\mathcal{H}$ as follows.

Definition 1 (Performance Metric [$\mathcal{H}$]) *Let i be a problem instance, and $\mathcal{A}_x(i)$ be the objective value of the corresponding solution for instance i obtained by a target algorithm $\mathcal{A}$ when executed*

[1] Singapore Management University, Singapore, email: lindawati.2008, hclau, fdzhu@smu.edu.sg

under configuration **x**. *Let $OPT(i)$ denote the best known value for instance i. $\mathcal{H}_x(i)$ is formulated as: $\mathcal{H}_x(i) = \frac{|OPT(i) - A_{\mathbf{x}}(i)|}{OPT(i)}$*

For benchmark instances with known global optimum value, we use the known global optimum value as its $OPT(i)$, while for new instances, we use the target algorithm's best solution. Using performance metric $\mathcal{H}$, we define the instance-specific parameter tuning problem as follows.

Definition 2 (Instance-Specific Parameter Tuning [ISPT]) *Given a set of instances I, a parameter configuration space Θ for a target algorithm $\mathcal{A}$ and a performance metric $\mathcal{H}$, the* ISPT *problem is to find a parameter configuration* **x** $\in \Theta$ *for each $i \in I$ such that $\mathcal{H}_x(i)$ is minimized over Θ.*

Instead of finding a parameter configuration for each problem instance, the ISPT problem can be approximated in a cluster-based manner in which problem instances are grouped into clusters and a parameter configuration is computed for each cluster [10, 12]. In this paper, we aim to solve the ISPT problem with user-specified constraints on the resulting clusters. In real-world applications, these user-specified constraints often represent domain knowledge to guide the clustering process. In particular, we consider a simple and natural set of constraints, each of which specifies whether a pair of data samples must belong to the same cluster, which are called *must-link* constraints (M_{link}), or must belong to different clusters, which are called *cannot-link* constraints (C_{link}) [3, 20].

Hence, we define the problem of constrained cluster-based instance-specific parameter tuning as follows.

Definition 3 (Constrained Cluster-based Instance-Specific Parameter Tuning [CC-ISPT]) *Given a set of instances I, a parameter configuration space Θ for a target algorithm $\mathcal{A}$, a performance metric $\mathcal{H}$, a set of user-specified constraints C, the* CC-ISPT *problem is to find a clustering π of all instances of I and a parameter configuration* **x** $\in \Theta$ *for each cluster of π such that (I) π satisfies all the constraints in C; and (II) the average $\mathcal{H}_x(i)$ for each cluster is minimized over Θ.*

If there is no user-specified constraints, user-specified constraints is considered as an empty set. We assume that there is no contradiction between M_{link} and C_{link}.

3 Related Work

Various approaches for *one-size-fits-all configurator* have been proposed in the literature which divided into two categories: *model-free* and *model-based*. For model-free approaches, F-Race and its variants [4] work by using statistical model selection to evaluate a set of candidate configurations and discarding statistically bad configurations. Other model-free approaches that can handle large number of parameters are ParamILS [9] and GGA [2]. ParamILS applies an iterated local search that uses an adaptive capping technique to speed up the search process. GGA (Gender-based Genetic Algorithm) uses a genetic algorithm which divides the parameter candidate set into two groups and applies a different selection method for each group. For model-based approaches, CALIBRA [1] combines statistical experimental design with local search, and this approach handles upto five parameters. A recent work SMAC [8] constructs predictive performance models to focus attention on promising regions of a design space. One common shortcoming of the above approaches is that they provide a **one-size-fits-all** configuration for all instances, which may not perform well on large and diverse instances.

Three recent approaches for instance-specific tuning is Hydra [19], ISAC [10] and CluPaTra [12]. Hydra works by combining automated parameter tuning and portfolio-based algorithm selection. It automatically builds a set of solvers with complementary strengths by iteratively configuring new algorithms to be used in its portfolio. ISAC and CluPaTra work by dividing instances into clusters based on feature(s) similarity and tuning the parameter configuration for each cluster. The main difference between ISAC and CluPaTra is that ISAC uses **problem-specific features** while CluPaTra uses a **generic feature**. For a given problem, ISAC uses different problem-specific features which require in-depth understanding of the problem. For example, in [10], 8 features are used for Set Covering Problem (SCP) and 9 features for Mixed Integer Programming Problem (MIP). CluPaTra uses a generic feature derived from search trajectories to perform clustering, and then tune the parameters for each cluster.

4 Solution Approach

Our approach follows the framework of CluPaTra [12] that makes use of the search trajectories. As CluPaTra uses sequence alignment to calculate similarity, it suffers from the following two limitations.

1. **Scalability.**
   In CluPaTra, pair-wise sequence alignment is implemented using standard dynamic programming with a complexity $O(m^2)$, where m is the maximum sequence length of the sequences. Hence, the total time complexity for all instances is $O(n^2 \times m^2)$, where n is the number of instances and m is the maximum sequence length. This poses a serious problem for instances with long search trajectories and when the number of instances is large.
2. **Flexibility.**
   The nature of sequence alignment is to align a pair of sequence segments that gives us the highest alignment score. A matched symbol contributes a positive score (+1), while a gap contributes a negative score (-1). The sum of the scores is taken as the maximal similarity score of the two sequences. However, it is possible that sequences share similarity on more than one segments, especially for long sequences. Sequence alignment is not flexible enough to capture multiple-segment alignment with an acceptable time complexity.

To overcome these limitations and achieve better overall performance, we propose a new algorithm called SufTra that uses a compact and rich data structure, namely Suffix Tree, and design a more efficient method to calculate similarity.

SufTra addresses CluPaTra's limitations as follows: (1) Scalability: We propose a linear time algorithm for both Suffix Tree construction and traversal; and (2) Flexibility: We generate compact patterns from search trajectories and use it as features. The patterns may occur in multiple segments along the search trajectory, so suffix trees enable us to consider multiple-segment similarities to improve the accuracy of the clusters.

We further improve the cluster quality by employing user supervision and proposing a new classification method to map testing instances to clusters. This method enables us to generate more accurate mapping in a shorter computation time. In the overall, SufTra runs in linear-time, which is an order of magnitude improvement and translates to a significantly faster method, compared to CluPaTra (that runs in quadratic time). Furthermore, we show experimentally that the quality of clusters outperform those of CluPaTra

4.1 SufTra Framework Overview

The SufTra framework works in two phases: training and testing. The training phase works as follows:

1. Feature Extraction. We extract a set of features F from search trajectories using a suffix tree and remove insignificant features with respect to M_{link} and C_{link}.
2. Similarity Calculation. We calculate similarity scores using the extracted features.
3. Clustering. We cluster the instances using AGNES (AGglomerative NESting).
4. Parameter Configuration. We run an existing one-size-fits-all configurator to obtain the best configuration for each cluster created.

In the testing phase, we use the knowledge from the training phase to return instance-specific configuration(s) for testing instances. This phase is usually performed online. To achieve this, we design a new method for fast and accurate testing instance mapping. Our proposed method consists of two steps:

1. Signature Construction. We construct the signatures for each cluster. This step is run once, and can be performed offline.
2. Cluster Mapping. For an arbitrary testing instance, we match its search trajectory to the cluster's signature and return parameter configuration from the best-matching cluster's as its parameter configuration. This step will be performed online.

Thus, our framework makes use four major components: feature extraction, similarity calculation, clustering and signature extraction. The details of these components are given as follows.

4.2 Feature Extraction

To generate its features, SufTra mines patterns from search trajectories. To mine compact and important patterns, we select pattern that has significant length and appear in a sufficient number of instances [6]. We also filter the patterns using user-specified constraints.Hence, we define the features as follows.

Definition 4 (Feature [$\mathcal{F}$]) *Let I be a set of problem instance, S be a set of search trajectories for all instance in I, min_{length} be a minimum length threshold, $min_{support}$ be a minimum support threshold, C be a set of user-specified constraints. $\mathcal{F}$ is defined as a set of distinct segments from S that: (I) has a length greater than min_{length}; (II) occurs in at least $min_{support}$ number of instances; and (III) has high score for discriminating between all pairs of instances with respect to C.*

The steps on Feature Extraction are as follows.

4.2.1 Search Trajectory Representation

Search trajectory is defined as a path of solutions discovered by the target algorithm $\mathcal{A}$ as it searches through the neighborhood search space [7]. We represent a search trajectory as a directed sequence of symbols before constructing suffix tree. The steps for converting a search trajectory into a string are as follows:

- Record the instance's search trajectory by running the target algorithm using a random parameter configuration.
- Convert the search trajectory to a sequence based on its solution's attributes. Each solution in search trajectory is represented as a symbol which encodes a combination of two solution attributes: (1) percentage deviation of quality from OPT (as defined in Definition 1); and (2) position type, based on the topology of the local neighborhood as given in Table 1 [7]. These two attributes are combined; of which the first two digits are the deviation of the solution quality and the last digit is the position type. To handle target algorithms with cycles and (random) restarts, it adds two additional symbols: 'C' and 'J'; 'C' is used when the target algorithm returns to a previously-visited position (cycle), while 'J' is used when the local search is restarted (jump). An example of a search trajectory sequence is *14L-14L-14L-14L-14L-14L-14L-14L-14L-14L-14L-14L-14L-04M-J-24L-14L-14L-14L-14L-14L-14L-14L*, where *14L* represent the solution that has percentage deviation 14% and position type LEDGE.

Table 1. Position Types of Solution

Position Type Label	Symbol	$<$	=	$>$
SLMIN (strict local min)	S	+	-	-
LMIN (local min)	M	+	+	-
IPLat (interior plateau)	I	-	+	-
SLOPE	P	+	-	+
LEDGE	L	+	+	+
LMAX (local max)	X	-	+	+
SLMAX (strict local max)	A	-	-	+

'+' = present, '-' = absent; referring to the presence of neighbor solutions with larger ('$<$'), equal ('=') and smaller ('$>$') objective values

- Construct a *Hash Table* to store number of repetitions. Note that in a search trajectory, several consecutive solutions may have similar solution properties before the final improvement and reaching local optimum. We therefore compress the search trajectory sequence to a *Hash String* by removing the consecutive repetition symbols and storing the number of repetitions in a *Hash Table* (to be used later in the similarity score calculation). Removing consecutive repetition symbols gives us two advantages: (1) it offers greater flexibility in capturing more varieties of similarity for symbol patterns between two instances. Two instances may share similar patterns but a different number of consecutive symbols, e.g., for a particular symbol $14L$, in one instance it may occur repeatedly for 10 times, while in the other instance it may occur repeatedly for 5 times. And (2) it reduces computational cost for constructing and traversing the suffix tree, since the time needed is decided by its length. *Hash String* is a more compact and shorter representation of the original search trajectory sequence. An example of *Hash String* is *14L-04M-J-24L-14L*.
- Convert the symbol for each solution to one single character and concatenate it into a string (*Hash String*).

4.2.2 Suffix Tree Construction

The suffix tree is a data structure that exposes the internal structure of a string for a particularly fast implementation of many important string operations. Suffix trees are used to solve exact and inexact matching problems in linear time and are widely used in substring problems [5]. The construction of a suffix tree proves to have a linear time complexity w.r.t. the input string length [5].

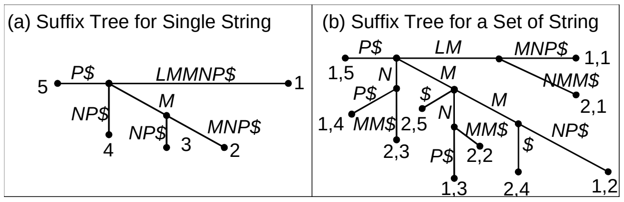


Figure 1. (a) Suffix Tree Example for string S_1 ($LMMNP$), number at a leaf indicates the starting position of the suffix in that string; (b) Generalized Suffix Tree Example for strings S_1=$LMMNP$ and S_2=$LMNMM$, the first number at a leaf indicates the string and the second number indicates the starting position of the suffix in that string.

A suffix tree T for a string S with m length is a rooted directed tree having exactly m leaves numbered 1 to m. Each internal node, other that the root, has at least two children and each edge is labeled with a substring (including the empty substring $) of S. No two edges out of a node has edge-labels beginning with the same character.

To represent suffixes of a set $\{S_1, S_2, ...S_n\}$ of strings, we use a $generalized$ suffix tree. $Generalized$ suffix tree is built by appending a different end of string marker (which is a symbol not in used in any of the strings) to each string in the set, then concatenate all the strings together, and build a suffix tree for the concatenated string [5]. An example of $generalized$ suffix tree for strings $LMMNP$ and $LMNMM$ is $LMMNPLMNMM$ The time to build this suffix tree is proportional to the total length of all the strings. The suffix tree for a single string S_1 and a set of string S_1 and S_2 is shown in Fig. 4.2.2.

In a suffix tree structure, we can easily retrieve matching substrings from a set of string by finding the branch that has leaves from the corresponding strings. From our suffix tree example (Fig. 4.2.2b), branches with edge-label M, N, LM, MM, and MN have leaves from both strings S_1 and S_2. Those edge-labels represent the same substring shared by S_1 and S_2.

We construct the suffix tree using Ukkonen's algorithm [5]. To cover every training instance, we build a single $generalized$ suffix tree for all the *Hash String*. The length of the concatenate string is proportional to the sum of all the *Hash String* lengths. Details of Ukkonen's algorithm can be found in [5]. Our implementation requires $O(n \times l)$, where n is the number of instances and l is the maximum length of the *Hash String*.

4.2.3 Frequent Substring Extraction

After constructing the suffix tree, we extract frequent substrings. A substring is considered as frequent if it has a length greater than $\min_{length}$ and it occurs in at least $\min_{support}$ number of strings [6]. The values of $\min_{length}$ and $\min_{support}$ are different for each problem. While fixing the values would sacrifice flexibility, it is also computationally challenging to find optimal values by exhaustive search. We therefore apply local search, a simple yet effective method to provide sufficiently good values in reasonable time.

We use local search to move from the initial values of $\min_{length}$ and $\min_{support}$ to their neighbors by changing either $\min_{length}$ or $\min_{support}$ at each move until the average distance among all instances in two different clusters is no longer improving.

To find the initial values of $\min_{length}$ and $\min_{support}$, we run a competition among 5 candidates, namely: lower bound of $\min_{length}$ and $\min_{support}$, upper bound of $\min_{length}$ and $\min_{support}$, mid value between lower and upper bound and two random values. Lower bound of $\min_{length}$ and $\min_{support}$ is set to 2, while upper bound is set to 20% of the number of instances for $\min_{length}$ and 20% of the maximum string length respectively for $\min_{support}$.

4.2.4 Feature Selection

We use user-specified constraints (*must-link* and *cannot-link constraints*) to select the "*good*" features from the frequent substrings. As in [20], we want to select features with the best constraint preserving ability. We assume that a "*good*" feature should appear more in instances that has *must-link* constraints rather than in instances that has *cannot-link* constraints.

Based on that assumption, we filter the substrings using the following steps:

1. Generate a set of cliques C for instances that belong to the same cluster based on M_{link}
2. Calculate discriminative score $DScore$ for each substring using the following formula:

$$DScore(s) = \frac{\sum_{k=0}^{|C|} \sum_{(i,j) \in C_k} (x(s)_{ij})}{\sum_{(i_i,i_j) \in C_{link}} (x(s)_{ij})} \quad (1)$$

where
$$x(s)_{ij} = \begin{cases} 1 & \text{if } s \text{ is appear in instance } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

3. Select n number of substring which has the highest score as features. The default value for n is $M/2$, where M is the number of substrings.

Note that, if the user-specified constraints is not available, we use all the extracted substrings as features.

4.3 Similarity Score Calculation

After having the features, we calculate the instance's score for each feature and construct an instance-feature metric using the following rules:

1. If the instance does not contain the feature, the score is 0.
2. Else the score is calculated by summing up the number of repetitions for each symbol in feature from previously constructed *Hash Table*. A frequent substring may occur multiple times in one string. We calculate the score for each occurrence and choose the maximum score as the score for the instance-feature metric.

With this metric, we calculate the similarity for each pair of instances by cosine similarity, a widely-used similarity measure for comparing vectors [5]. Cosine similarity is formulated as:

$$similarity = \frac{\sum_{i=0}^{n} (f_i(I_1) \times f_i(I_2))}{\sqrt{\sum_{i=0}^{n} f_i(I_1)^2} + \sqrt{\sum_{i=0}^{n} f_i(I_2)^2}} \quad (2)$$

where $f_i(I_1)$ and $f_i(I_2)$ are the scores from the instance-feature metric for Instance 1 and 2 respectively.

4.4 Clustering

Similar to [12], we cluster the instances by a well-known clustering approach AGNES [6] with L method [16]. AGNES or AGglomerative NESting is a hierarchical clustering approach that works by creating clusters for each individual instance and then merging two closest clusters (i.e., a pair of clusters with the smallest distance) resulting in fewer clusters of larger sizes until all instances belong to one cluster or a termination condition is satisfied (e.g. a prescribed number of clusters is reached). We implement the L method [16] to automatically find the optimal number of clusters, which works by using the evaluation graph where the *x*-axis is the number of clusters and the *y*-axis is the value of the evaluation function at *x* clusters. In this paper, we use the average distance among all instances in two different clusters as the evaluation function. L method fits the curve in the evaluation graph into two lines and chooses the intersection point between these two lines as the optimal number of clusters.

4.5 Signature Extraction

In testing phase, we use signatures to represent instance's search trajectories in each cluster. We define signature as follows.

Definition 5 (Signature $\mathcal{SIGN}$) *Let c be a cluster of problem instance i, S be a set of search trajectories for all instance i in c, min_{length} be a minimum length and $min_{support}$ be a minimum support. $\mathcal{SIGN}(c)$ is defined as a set of distinct segments from S that has significant length (greater than min_{length}) and appears in most of instance i in c (at least $min_{support}$ number of instances).*

The steps for signature extraction are similar to feature extraction (Section 4.2), but for each cluster, we need to construct a different suffix tree and extract the features from each suffix tree. Since each cluster already satisfy user-specified constraints, we skip the feature selection process and use all the extracted substrings as features. Unless stated otherwise, min_{length} and $min_{support}$ are set to min_{length} and $min_{support}$ value in feature extraction.

4.6 Time Complexity

The time complexity for generating *Hash String* is $O(n \times m)$ with n being the number of instances and m being the maximum string length, while that for constructing the suffix tree is $O(n \times l)$ with n being the number of instances and l being the maximum *Hash String* length. Extracting features, building instance-feature metric and calculating similarity for each pair of instances can be done by a single traversal of the suffix tree structure. Hence it requires $O(n \times l)$ with n being the number of instances and l being the maximum *Hash String* length. The clustering process requires $O(n^2)$ with n being the number of instances. Hence, the overall time complexity for the training phase, excluding the time needed for tuning, is $O(n^2 + (n \times m) + (n \times l))$ with n being the number of instances, m being the maximum string length and l being the maximum *Hash String* length ($n \ll l \ll m$).

In the testing phase, cluster signature construction requires $O(n \times l)$ where n is the number of training instances and l is the maximum *Hash String* length. For a given testing instance, we match the signatures with the testing instance's strings, which takes $O(signature_c \times l_{signature} \times m \times l)$ time, where $signature_c$ is the total number of signatures in all clusters, $l_{signature}$ is the maximum length of signatures, m is the number of testing instances, and l is the maximum string length of the testing instance ($l_{signature} \ll l$).

5 Empirical Evaluation

In this section, we present our experimental results to measure the efficiency and effectiveness of SufTra on a classical COPs, Quadratic Assignment Problem (QAP).We compare our approach with two other instance-specific configurators in the literature: ISAC [10] and CluPaTra [12]. Note that since our aim is to measure solution quality, we do not compare our approach with Hydra [19], another instance-specific configurator that seeks to optimize run time performance but not solution quality of the target algorithm.

As a target algorithm, we use hybrid Simulated Annealing and Tabu Search (SA-TS) algorithm [13], which uses the Greedy Randomized Adaptive Search Procedure (GRASP) to obtain an initial solution, and a combined Simulated Annealing (SA) and Tabu Search (TS) algorithm to improve it. There are four parameters to be tuned as described in Table. 2.

Table 2. Parameters for SA-TS on QAP

Parameter	Description	Type	Range
Temp	Initial temperature of SA algorithm	Continuous	[100, 5000]
Alpha	Cooling factor	Continuous	[0.1, 0.9]
Length	Length of tabu list	Discrete	[1, 10]
Pct	Percentage of non-improving iterations prior to intensification strategy	Continuous	[0.01, 0.1]

We used two set of instances: SET A and SET B as described in Table. 3. All experiments were performed on a 1.7GHz Pentium-4 machine running Windows XP.

Table 3. Set of Instances for QAP

Set	Description	n_{tr}	n_t	m
SET A	benchmark instances from QAPLib with number of city 22 to 150	40	10	4,613
SET B	generated instances from two generators as in [14] with number of facilities varied from 5 to 150	100	400	15,536

5.1 Cluster Analyses

We experimented on SET A instances and compared the clusters created from SufTra, CluPaTra and ISAC. Since ISAC requires problem-specific features, we used 2 features: *flow dominance* and *sparsity* of flow metric [17].

We measured the cluster quality using *extrinsic* method. *Extrinsic* methods compare the clusters against the known class labels or *ground-truth* clusters (i.e. the set of clusters which is supposed to represent the ideal/optimal clustering) [6]. We used the existing well-studied classification of QAP instances based on the distance and flow metrics, due to [18] as *ground truth* cluster.

For this experiment, we used two different SufTra implementation. For SufTra, we did not include user-specified constraints information, while for SufTra(c), we randomly derived 20 user-specified constraints from *ground truth* cluster.

The cluster quality values are shown in Table. 4 (I). Notice that SufTra(c) and SufTra has higher cluster quality compared to CluPaTra and ISAC.

5.2 Performance Comparison

To evaluate SufTra's effectiveness for long search trajectories and large number of instances, we ran experiment on SET B. For SufTra(c), we randomly used 20 user-specified constraints where we derived from the following assumption: (1) instances from the same generator are considered on the same cluster; and (2) instances from different generators are considered on different clusters.

Table 4. QAP Experiment Result

	Training	Testing
I. Clustering Analyses		
CluPaTra	0.68	0.70
ISAC	0.80	0.80
SufTra	0.85	0.85
SufTra(c)	**0.91**	**0.93**
II. Computational Time		
CluPaTra	1,051 s	2,718 s
SufTra	**56 s**	**146 s**
SufTra(c)	65 s	147 s
III. Performance Result		
ParamILS	1.08	2.14
CluPaTra	0.89	1.58
ISAC	0.84	1.22
SufTra	0.83	1.16
SufTra(c)	**0.80**	**1.15**
p-value**	0.061	0.042

**based on statistical test on ISAC and SufTra

First, we compared the time needed (in seconds) for SufTra, SufTra(c) and CluPaTra to form the clusters in training phase and to map the testing instances in testing phase. Table. 4 (II) shows the result. From the table, we observe that SufTra and SufTra(c) is 18 times faster then CluPaTra.

Next, we compared the target algorithm performance using parameter configuration from SufTra, SufTra(c), CluPaTra and ISAC as well as the one-size-fits-all configurator ParamILS. For the three instance-specific methods, we used the same one-size-fits-all configurator, ParamILS [9]. Since ParamILS works only with discrete parameters, we first discretized the values of the parameters. We measured the performance using performance metric as defined in Definition 1.

We set the cutoff runtime of ParamILS to 100 second. For CluPaTra, SufTra and SufTra(c), we allowed each configurator to execute the target algorithm for a maximum of two CPU hours for each cluster. To ensure fair comparison, we set the time budget for ISAC and ParamILS to be equal to the total time needed to run SufTra. For unbiased evaluation, we used a 5-fold cross-validation [6] and measured the average performance over all folds. We also performed a statistical test (*t-test*) on the significance of our result where a *p-value* below 0.05 is deemed to be statistically significant. In Table. 4 (III), we show the performance comparison results. From the table, we observe that SufTra and SufTra(c) performs better on training and testing instances compare to other approaches. But the result for training instances is not statistically significant compare to ISAC.

6 Conclusion and Future Works

In this paper, we proposed a new generic instance-based configurator via the clustering of patterns according to the instance search trajectories using the suffix tree data structure and user specified-constraints. We measured the cluster quality and performance of QAP, and show that SufTra result (with or without user-specified constraints) is outperform the existing methods. Its cluster is almost similar to existing benchmark instance classifications, thereby showing its effectiveness. Hence, we claim that: (1) SufTra (with or without user-specified constraints) is a suitable approach for instance-specific configuration that significantly improves the performance with minor additional computational time; and (2) SufTra has overcome CluPaTra limitations with a new efficient method for feature extraction and similarity computation using suffix tree.

Up to this stage of our work, the SufTra framework can only be applied to target algorithms which are local-search-based, since our approach uses search trajectory as the generic feature. As future works, we will investigate how to generate clusters from population-based-algorithm using generic features pertaining to population dynamics.

REFERENCES

[1] Belarmino Adenso-Díaz and Manuel Laguna, 'Fine-tuning of algorithms using fractional experimental design and local search', *Operations Research*, **54**(1), 99–114, (2006).

[2] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney, 'A gender-based genetic algorithm for the automatic configuration of algorithms', in *15th international Conference on Principles and Practice of Constraint Programming*, pp. 142–157, (2009).

[3] Sugato Basu, Mikhail Bilenko, and Raymond J. Mooney, 'A probabilistic framework for semi-supervised clustering', in *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, pp. 59–68, (2004).

[4] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, 'Automated algorithm tuning using f-races: Recent development', in *8th Metaheuristics International Conference*, (2009).

[5] Dan Gusfield, *Algorithms on Strings, Trees and Sequences*, Cambridge University Press, 1997.

[6] J. Han and M. Kamber, *Data Mining: Concept and Techniques, 2nd Edition*, Morgan Kaufman, San Francisco, 2006.

[7] H.H. Hoos and T. Stützle, *Stochastic Local Search: Foundation and Application*, Morgan Kaufman, San Francisco, 2004.

[8] F. Hutter, H.H. Hoos, and K. Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *LNCS: 5nd Learning and Intelligent OptimizatioN Conference*, (2011).

[9] F. Hutter, H.H. Hoos, K. Leyton-Brown, and T. Stützle, 'Paramils: An automatic algorithm configuration framework', *Journal of Artificial Intelligence Research*, **36**, 267–306, (2009).

[10] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, 'Isac:instance-specific algorithm configuration', in *19th European Conference on Artificial Intelligence*, (2010).

[11] G. Klau, N. Lesh, J. Marks, and M. Mitzenmacher, 'Human-guided tabu search', in *In: National Conference on Artificial Intelligence (AAAI)*, (2002).

[12] Lindawati, Hoong Chuin Lau, and David Lo, 'Instance-based parameter tuning via search trajectory similarity clustering', in *LNCS: 5nd Learning and Intelligent OptimizatioN Conference*, (2011).

[13] K.M. Ng, A. Gunawan, and K.L. Poh, 'A hybrid algorithm for the quadratic assignment problem', in *International Conf. on Scientific Computing*, pp. 14–17, (2008).

[14] Gabriela Ochoa, Sebastien Verel, Fabio Daolio, and Marco Tomassini, 'Clustering of local optima in combinatorial fitness landscape', in *LNCS: 5nd Learning and Intelligent OptimizatioN Conference*, (2011).

[15] C.R. Reeves, 'Landscapes, operators and heuristic search', *Annals of Operations Research*, **86**(1), 473–490, (1999).

[16] S. Salvador and P. Chan, 'Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms', in *16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 576–584, (2004).

[17] T. Stützle and S. Fernandes, 'New benchmark instances for the qap and the experimental analysis of algorithms', in *LNCS: Evolutionary Computation In Combinatorial Optimization*, (2004).

[18] É.D. Taillard, 'Comparison of iterative searches for the quadratic assignment problem', *Location Science*, **3**(2), 87–105, (1995).

[19] L. Xu, H.H. Hoos, and K. Leyton-Brown, 'Hydra: Automatically configuring algorithms for portfolio-based selection', in *Conference of the Association for the Advancement of Artificial Intelligence (AAAI-10)*, (2010).

[20] Daoqiang Zhang, Songcan Chen, and Zhi-Hua Zhou, 'Constraint score: A new filter method for feature selection with pairwise constraints', *Pattern Recognition*, **41**(5), 1440–1451, (2008).

Learning-to-rank with Prior Knowledge as Global Constraints

Tiziano Papini and **Michelangelo Diligenti** [1]

Abstract. A good ranking function is the core of any Information Retrieval system. The ranking function can be a simple cosine similarity or, more likely for any advanced IR system like a Web search engine, a function processing a high number of signals which typically requires a lot of hand-tuning to deal with the large variability of queries. The result is a sub-optimal and highly complex ranking function which, in spite of having been crafted by human experts, is hard to control and debug. In the last few years, learning-to-rank from examples has emerged as a more flexible approach to design ranking functions. While learning-to-rank approaches have been proved to significantly outperform hand-tuned solutions, they still feature many disadvantages. First, they rely on a large number of training examples to model the high variability of the input query stream. Unfortunately, constructing a training set is more complicated than labeling examples in classical supervised classification tasks. Indeed, the labeling process for learning-to-rank tasks is inherently error-prone and incomplete. Secondly, learning-to-rank schemas usually do not account for the explicit knowledge that human experts have built over the years. It would be nice to integrate this knowledge without having to rely on a large set of examples to infer it. Finally, the proposed approach opens new ways to integrate unlabeled data into the learning process, as the rules must be respected also by the unlabeled data. This paper presents a general framework to convert prior knowledge in form of First Order Logic (FOL) clauses into a set of continuous constrains and shows how these constraints can be integrated into any learning-to-rank approach which is optimized via gradient descent.

1 Introduction

Modern Information Retrieval systems like Web search engines have available hundreds of features to represent each document to answer users' queries. These features range from query-independent signals like PageRank to others measuring the match between the query and a document. Human experts can compose these signals in order to get a ranking function which is effective in most cases. However, because of the high number of correlated signals, it is hard to manually design ranking functions achieving results that are close to optimality. Learning-to-rank from examples has emerged as a more flexible approach to design ranking functions, which has gained popularity both in research and industrial contexts. Learning-to-rank approaches have been proved to significantly outperform hand-tuned solutions [1], but they still feature many disadvantages. First, they rely on a large number of training examples to model the high variability of the input query stream, in a context where the labeling process is inherently error-prone, leading to inconsistent, incomplete and noisy training sets. Secondly, learning-to-rank schemas usually discard the explicit knowledge that human experts have built over several years, starting the optimization process from scratch.

The basic assumption of this paper is that an expert can express the desired behavior of the ranking function via a set of FOL clauses, and it is required to integrate this knowledge without having to rely on a large set of examples to infer it. The training examples will also respect the rules and a supporter of pure learning-to-rank approaches could claim that the rules can be inferred from the examples. However, the number of training examples is often not sufficient to converge to the optimal ranking function and it is therefore more compact and effective to express the ranking rules in an explicit form. Another advantage of this approach is that it allows to get advantage of the unlabeled data into the learning process, as the data can be used to ensure that the constraints are enforced. In a learning-to-rank setting, unlabeled data means the actual ranking provided by the IR system, with no labeling nor goodness judgments assigned by a human expert. Therefore, unlabeled data can be obtained in a virtually unlimited quantity with little cost.

This paper presents a general framework to encode arbitrary prior knowledge expressed as First Order Logic (FOL) into a set of continuous constraints, which can be added to the cost function to optimize. For example, a constraint could force some degree of diversity or freshness in the results, or make the ranking to be determined by a single feature under a specific condition. The presented framework is not tied to a specific learning-to-rank approach, but it can be integrated into any technique, which is optimized via gradient descent. In particular, the experimental section employs a variant of *LambdaRank*, called *LambdaNeuralRank*, to show the effectiveness of the proposed technique. The experimental results show that the approach improves state-of-the-art results on a dataset built from the logs of a commercial search engine when little supervised data is available.

2 Prior work

Learning-to-rank approaches can be grouped into three main classes: pointwise, pairwise and listwise approaches. A pointwise approach [2] takes document/score pairs as training examples to learn a document scoring function $f : \mathcal{D} \Rightarrow I\!R$, which assigns a score to a document, under the assumption that the final rank of the documents will be obtained by sorting the documents in descending order of score. Pairwise methods like [3] take a set of pairs of documents as input to the training. The training process consists in learning a function, which correctly orders the pairs ($f(u) > f(v)$ if u should precede in the rank v). Pairwise approaches are generally preferred to pointwise methods, because they do not impose specific scores to the learning algorithm, leaving it the freedom to select the score range in which to work. Finally, listwise methods [4, 5] get a set of lists of ranked

[1] Dipartimento di Ingegneria dell'Informazione, Università di Siena, Italy, email: {papinit,diligmic}@dii.unisi.it

documents as training examples, and the optimization is performed using a loss function over the entire list of documents.

A recent trend in learning to rank approaches is to attempt a direct optimization of the target metrics [6][7], which typically are either *Mean Average Precision* (MAP) for ranks having two relevance scores or *Normalized Discounted Cumulative Gain* (NDCG) when there is an arbitrary number of relevance scores. This class of approaches falls in the listwise category as the target metrics are functions of ranked lists and not of individual pairs. Those approaches are generally considered to outperform pairwise methods as they can direct the learning toward what's most important with regard to the optimization of the target metric. However, direct optimization of the target metrics is difficult, because all the commonly employed metrics, such as NDCG and MAP, are not expressed in terms of the scoring functions but in terms of the document ranks (which then depend on the functions). This makes the resulting loss function either constant or not differentiable in any point with respect of the training parameters. Most learning approaches solve this issue by employing a continuous approximation of the target metric [8].

3 LambdaRank, LambdaNeuralRank and constraints

The proposed methodology to inject prior knowledge can be integrated into any learning-to-rank approach trained via gradient descent. We start introducing a variant of LambdaRank, a state-of-the-art learning-to-rank algorithm, which will be used as underlying learning mechanism over which FOL knowledge is injected.

Let $C(f, \boldsymbol{d})$ be the following cost function

$$C(f,\boldsymbol{d}) = \underbrace{\sum_{i,j\in\boldsymbol{d},i>j} L\left(f(d_i), f(d_j)\right)}_{\mathcal{L}(f,\boldsymbol{d})} + \underbrace{\frac{\lambda_r}{2}||\boldsymbol{w}||^2}_{\mathcal{R}(f)} + \underbrace{\sum_{h=1}^{H}\lambda_c^h\Phi_h(f)}_{\mathcal{C}(f)}$$

where $[f(d_1), \cdots, f(d_n)]$ is the vector of scores assigned by f to the set of documents to score $\boldsymbol{d}$, $i > j$ indicates that document d_i should be ranked higher than d_j , $\boldsymbol{w}$ is the vector of parameters of f, λ_r determines the trade-off between fitting the supervised data and the magnitude of the weights, L is a loss function and the H logic clauses forming the prior knowledge are represented as a set of H constraints $\Phi_h(f)$, each assigned a weight λ_c^h. $\mathcal{L}(f,\boldsymbol{d}), \mathcal{R}(f), \mathcal{C}(f)$ indicate the labeled, regularization and constraint contributions to the cost function, respectively.

In particular, L was selected to be a hinge function shifted by a value ϵ which acts as the desired margin, yielding,

$$\begin{aligned} C(f,\boldsymbol{d}) &= \sum_{i,j\in\boldsymbol{d},i>j} h\left(f(d_i) - f(d_j) - \epsilon\right) + \\ &+ \frac{\lambda_r}{2}||\boldsymbol{w}||^2 + \sum_{h=1}^{H}\lambda_c^h\Phi_h(f) . \end{aligned} \tag{1}$$

When $\lambda_c^h = 0, h = 0, \ldots, H$, this cost function is equivalent to the cost function used by SVMRank [9] (in the primal). The derivative of the cost function with respect of a generic weight of f is the following:

$$\begin{aligned} \frac{\partial C(f,\boldsymbol{d})}{\partial w} &= \frac{\partial \mathcal{R}(f)}{\partial w} + \frac{\partial \mathcal{C}(f)}{\partial w} + \frac{\partial \mathcal{L}(f,\boldsymbol{d})}{\partial w} = \\ &= \lambda_r w + \sum_{h=1}^{H}\lambda_c^h\frac{\partial \Phi_h(f)}{\partial w} + \\ &+ \sum_{i,j\in\boldsymbol{d},i>j} \underbrace{h'\cdot\left(\left.\frac{\partial f}{\partial w}\right|_{d_i} - \left.\frac{\partial f}{\partial w}\right|_{d_j}\right)}_{\lambda_{ij}} , \end{aligned} \tag{2}$$

where λ_{ij} is the contribution to the derivative of a single pair of documents.

LambdaRank [10] provides a learning-to-rank basic mechanism, which acts as a meta-methodology to optimize a target metric. LambdaRank is a meta-approach as it relies on an underlying pairwise algorithm (like the one presented so far) for computing the gradients for each pair. LambdaRank transforms a pairwise approach into a listwise one by weighting the contribution to the gradient of each pair as shown in equation 2 by its utility in improving the target metric. Regardless of the exact formulation of the cost function optimized, LambdaRank has been proved to locally maximize the target metric [11]. This is theoretically justified by observing that it is not needed to explicitly know the cost function we are optimizing, only the gradients are needed.

In particular, let $\theta(\boldsymbol{d}, \mathbf{rnk})$ be the goodness score of rank $\mathbf{rnk}$ over documents $\boldsymbol{d}$ and $\Delta\theta(\boldsymbol{d}, \mathbf{rnk}, i, j)$ be the difference of the θ values coming from swapping document i and j of $\boldsymbol{d}$ in $\mathbf{rnk}$. LambdaRank weights the the contributions λ_{ij} by the impact on the target metric $\Delta\theta(\boldsymbol{d}, \mathbf{rnk}, i, j)$. Therefore, the derivative of the labeled part with respect to a weight w assumes the following form,

$$\frac{\partial \mathcal{L}(f,\boldsymbol{d})}{\partial w} = \sum_{i,j\in\boldsymbol{d},i>j} \lambda_{ij}\cdot\Delta\theta(\boldsymbol{d}, \mathbf{rnk}, i, j) . \tag{3}$$

This value can be directly used to optimize the function by gradient descent together with $\frac{\partial \mathcal{R}(f)}{\partial w}$ and $\frac{\partial \mathcal{C}(f)}{\partial w}$, which we will show in the following section how to compute.

A common choice for the target metric in learning-to-rank environments is the NDCG, defined as,

$$NDCG(\boldsymbol{d}, \mathbf{rnk}) = \sum_{j=1}^{|\boldsymbol{d}|}\frac{2^{scr(d_j)} - 1}{\log_2(rnk(d_j)) + 1} .$$

where $scr(d_j)$ and $rnk(d_j)$ are the relevance score and its rank in the result set for document d_j.

It is easy to show that for NDCG, $\Delta NDCG(Bd, \mathbf{rnk}, i, j))$ is equal to,

$$\begin{aligned} \Delta NDCG(\boldsymbol{d}, \mathbf{rnk}, i, j) &= \frac{(2^{scr(d_i)} - 2^{scr(d_j)})}{\log_2(rnk(d_j) + 1)} - \\ &- \frac{(2^{scr(d_j)} - 2^{scr(d_i)})}{\log_2(rnk(d_i) + 1)} . \end{aligned} \tag{4}$$

Equation 4 can be used to replace $\Delta\theta(\boldsymbol{d}, \mathbf{rnk}, i, j)$ in equation 3. The function f can be implemented using any machine learning approach. In our experimental setting, f was implemented by a Neural Network as in RankNet [3]. For this reason, we called this learning-to-rank approach *LambdaNeuralRank*.

4 Prior Knowledge expressed as FOL and its conversion

In order to implement learning-from-constraints as described in the previous section, one needs to devise a conversion process to translate logic formalisms into real-valued functions. We focus our attention on knowledge-based descriptions given by first-order logic (FOL–KB). While the framework can be easily extended to arbitrary FOL predicates, in this paper we will consider only unary predicates to keep the notation simple.

Any FOL clause has an equivalent version in *Prenex Normal form* (PNF), that has all the quantifiers ($\forall, \exists$) and their associated quantified variables at the beginning of the clause. Standard methods exist to convert a generic FOL clause into its corresponding PNF and the conversion can be easily automated. Therefore, without loss of generality, we restrict our attention to FOL clauses in the PNF form. In the following, we indicate by $\mathcal{V} = \{v_1, \dots, v_N\}$ the set of the variables used in the KB. In particular, v_i can take values over the document collection $\mathcal{D}$ to be ranked or the set of queries $\mathcal{Q}$. Predicates can be of two types. The first type takes a document and returns $true$ if the document/query meets some predefined requirement (like having a feature above a value or its content belongs to some topical category, etc.). This first set of predicates used in the KB is indicated as $\mathcal{P} = \{p_k | p_k : (v \in \mathcal{D}) \rightarrow \{true, false\},\ k = 1, \dots, |P|\}$. A second class of predicates $\mathcal{R}$ (*ranking predicates*) checks whether a document d is in the first i positions of a rank over a set of n documents $\boldsymbol{d} = \{d_1, \cdots, d_n\}$, as determined by the ranking function f for a query q. Let $\mathcal{R} = \{r_k | r_k : (q \in \mathcal{Q}, v \in \mathcal{D}, I\!R^n) \rightarrow \{true, false\},\ k = 1, \dots, |R|\}$. The clauses will be built from the set of atoms $p(v) : (p \in \mathcal{P}, v \in \mathcal{D}) \vee (p \in \mathcal{R}, v \in \mathcal{D} \times \mathcal{Q} \times I\!R^n)$. Predicates $\mathcal{P}$ are given and they only select whether a document has some desired characteristic. On the other hand, predicates $\mathcal{R}$ depend on the ranking functions, which determine the rank of a document for a given query.

We assume that all the clauses are in the following general form:

$$\forall q \in \mathcal{Q}\ \ \{\forall, \exists\} v_1\ \dots\ \{\forall, \exists\} v_n\ \ E(q, p(v_1), \dots, p(v_n))$$

where $E(q, p(v_1), \dots, p(v_n))$ is a generic propositional expression over the atoms.

The FOL–KB will contain a set of clauses corresponding to expressions with no free variables (i.e. all the variables appearing in the expression are quantified) that are assumed to be *true* in the considered domain. These clauses can be converted into a set of constraints that can be enforced during the learning process. As detailed in the following sections, the conversion process of a clause into a constraint functional consists of the following steps: *Conversion of the ranking predicates*, *Conversion of the Propositional Expression* and *Quantifier conversion*.

4.1 Conversion of the ranking predicates

A ranking predicate $r_k(\boldsymbol{d}, d, q)$ can be implemented by computing $\boldsymbol{f}$ over $\boldsymbol{d}$ for q, then by sorting the scores and, finally, by checking that the rank condition on the target document is verified. The issue with this ranking predicate is that it is either constant or not differentiable in any point with respect of the parameters of the ranking function f. This makes optimization with respect of the parameters difficult. As commonly done in the context of learning to rank [8], it is possible to approximate the ranking predicate with a continuous and differentiable surrogate approximation of the predicate. This can be done by defining the expected position of d in the rank established by f over the set of document $\boldsymbol{d}$ as:

$$\hat{pos}(\boldsymbol{d}, d, f) = \sum_{i=1}^{|\boldsymbol{d}|} P(d_i > d) = \sum_{i=1}^{|\boldsymbol{d}|} \frac{e^{-\alpha[f(d) - f(d_i)]}}{1 + e^{-\alpha[f(d) - f(d_i)]}}$$

where $P(d_i > d)$ models the probability that d_i precedes d using a sigmoidal function and $\alpha > 0$ is a constant. The predicate $r_k(\boldsymbol{d}, d, q)$ can now be approximated by using a sigmoidal function on top of the estimate of the rank

$$\hat{r_k}(\boldsymbol{d}, d, q) = sigm(k - \hat{pos}(\boldsymbol{d}, d, f))$$

.

4.2 Conversion of the Propositional Expression

Prior work in the literature [12] concentrated on conversion schema based on t-norms. A different approach based on mixtures of Gaussians is exploited in this paper. This approach has been inspired by the methodology described in [13] to integrate symbolic knowledge into neural networks. The advantage of this approach relies in the fact that it does not perform any independence assumption among the variables, like it happens for t-norms.

In particular, let us consider a propositional logic clause involving n logic variables. The logic clause is equivalent to its truth table containing 2^n rows, each one corresponding to a configuration of the variables. The continuous function approximating the clause is based on a set of Gaussian functions, each one centered on a configuration corresponding to the true value in the truth table. The mixture function generalizes the propositional clause into a continuous setting by summing up the single Gaussians:

$$\begin{aligned} & t(x_1, \dots, x_n) = \\ & = \sum_{[c_1, \dots, c_n] \in \mathcal{T}} \exp\left(- \frac{||[x_1, \dots, x_n]' - [c_1, \dots, c_n]'||^2}{2\sigma^2} \right) , \end{aligned} \qquad (5)$$

where $x_1, \dots, x_n$ is the set of variables in the propositional clause and $\mathcal{T}$ is the set of all possible configurations of the input variables which correspond to the true value in the table.

For example, let us consider the clause $x \vee y$, which is verified by the three configurations $[true, true]$, $[true, false]$, and $[false, true]$. The clause is converted as $t(x, y) = \exp\left(-\frac{||[x,y]' - [1,1]'||^2}{2\sigma^2}\right) + \exp\left(-\frac{||[x,y]' - [1,0]'||^2}{2\sigma^2}\right) + \exp\left(-\frac{||[x,y]' - [0,1]'||^2}{2\sigma^2}\right)$.

It is also possible to generalize the clause by setting the default value to true and modeling the false configurations in the truth table:

$$\begin{aligned} & t(x_1, \dots, x_n) = \\ & = 1 - \sum_{[c_1, \dots, c_n] \in \mathcal{F}} \exp\left(- \frac{||[x_1, \dots, x_n]' - [c_1, \dots, c_n]'||^2}{2\sigma^2} \right) , \end{aligned}$$

where $\mathcal{F}$ is the set of all possible configurations of the input variables which correspond to a false value in the table.

If $[x, y]'$ assumes values corresponding to a configuration verifying the clause, $t(x, y) \geq 1$ holds. For a generic input $[x, y]'$, the value of $t(x, y)$ will decrease depending on the distance from the closest configuration verifying the clause. The variance σ^2 is a parameter that can be used to determine how quickly $t(x, y)$ decreases when moving away from a configuration verifying the constraint. Please note that each configuration verifying the constraint is always a global maximum of t.

4.3 Quantifier conversion

The quantified portion of the expression is processed recursively by moving backward from the inner quantifier in the PNF expansion.

Let us consider the universal quantifier first. The universal quantifier expresses the fact that the expression must hold for any result in the set of documents to rank $\boldsymbol{d}$. When considering the real–valued mapping of the original boolean expression, the universal quantifier can be naturally converted measuring the degree of non-satisfaction of the expression over the domain $\boldsymbol{d}$. More generally, let $\boldsymbol{v}_E$ be the vector of variables contained in the expression E, the satisfaction measure can be implemented by computing the overall distance of the penalty associated with E, i.e. $\varphi_E(\boldsymbol{v}_E, f)$ from the constant function equal to 0 over the domain $\boldsymbol{d}$. Using the infinity norm on discrete domains, this measure is

$$\forall v_q \in \boldsymbol{d} \;\; E(\boldsymbol{v}_E, \mathcal{P}) \rightarrow \max_{v_q \in \boldsymbol{d}} |\varphi_E(\boldsymbol{v}_E, \mathcal{P})| \; , \tag{6}$$

where the resulting expression depends on all the variables in $\boldsymbol{v}_E$ except v_q, and $\varphi_E(\boldsymbol{v}_E, \mathcal{P}) = max(1 - t_E(\boldsymbol{v}_E, \mathcal{P}), 0)$. Hence, the result of the conversion applied to the expression $E_q(\boldsymbol{v}_{E_q}, \mathcal{P}) = (\forall v_q \in \boldsymbol{d} \quad E(\boldsymbol{v}_E, \mathcal{P}))$ is a functional $\varphi_{E_q}(\boldsymbol{v}_{E_q}, \mathcal{P})$, assuming values in $[0, 1]$ and depending on the set of variables $\boldsymbol{v}_{E_q} = [v_{s(1,E_q)}, \ldots, v_{s(n_{E_q},E_q)}]$, such that $n_{E_q} = n_E - 1$ and $v_{s(j,E_q)} \in \{v_r \in \mathcal{V} | \exists i \;\; v_r = v_{s(i,E)}, \; v_r \neq v_q\}$. The variables in $\boldsymbol{v}_{E_q}$ need to be quantified or assigned a specific value in order to obtain a constraint functional depending only on the functions $\mathcal{P}$.

If we consider the conversion of the PNF representing a FOL constraint without free variables, the variables are recursively quantified until the set of the free variables is empty. In the case of the universal quantifier we apply again the mapping described previously. The existential quantifier can be realized by enforcing the De Morgan law ($\exists v_q \in \boldsymbol{d} \;\; E(\boldsymbol{v}_E, \mathcal{P}) \Longleftrightarrow \neg \forall v_q \in \boldsymbol{d} \;\; \neg E(\boldsymbol{v}_E, \mathcal{P})$) to hold also in the continuous mapped domain. Using the conversion of the universal quantifier of equation (6), we obtain the following conversion for the existential quantifier

$$\exists v_q \in \boldsymbol{d} \;\; E(\boldsymbol{v}_E, \mathcal{P}) \quad \rightarrow \quad \min_{v_q \in \boldsymbol{d}} |\varphi_E(\boldsymbol{v}_E, \mathcal{P})| \; .$$

It is also possible to select a different norm on the discrete domain to convert the universal quantifier. For example, when using the $\| \cdot \|_1$ norm for discrete domains, yields the conversion rule

$$\forall v_q \in \boldsymbol{d} \;\; E(\boldsymbol{v}_E, \mathcal{P}) \rightarrow \frac{1}{|\boldsymbol{d}|} \sum_{v_q \in \boldsymbol{d}} |\varphi_E(\boldsymbol{v}_E, \mathcal{P})| \; .$$

As an example of the conversion procedure, let q be a query and $s(\cdot), n(\cdot)$ two predicates checking whether a result is about topic $sport$ or $nutrition$, respectively. The clause $\forall v \in \boldsymbol{d} \;\; r_k(\boldsymbol{d}, v, q) \Rightarrow s(v) \vee n(v)$ expresses the fact that any result in the first k positions of results for q must be about topic $sport$ or $nutrition$. The only false configuration for the quantifier-free expression is $r_k(\boldsymbol{d}, v, q) = true, \; s(v) = false, \; n(v) = false$. Therefore, $t(q, v) = 1 - \exp\left(- \frac{(r_k(\boldsymbol{d},v,q)-1)^2 + s(v)^2 + n(v)^2}{2\sigma^2}\right)$. Then, converting the quantifier, adding the loss and inserting the continuous approximation of the rank predicate, we get the following constraint:

$$\begin{aligned} \Phi(f) &= \max_{v \in \boldsymbol{d}} \\ &\exp\left(- \frac{\left(sigm\left(k - \sum_{i=1}^{|\boldsymbol{d}|} \frac{e^{-\alpha[f(d)-f(d_i)]}}{1+e^{-\alpha[f(d)-f(d_i)]}} \right) - 1 \right)^2 + s(v)^2 + n(v)^2}{2\sigma^2} \right) = \\ &= 0 \; . \end{aligned}$$

5 Experimental Results

Unfortunately, publicly available learning-to-rank datasets like LETOR[2] are not suited for testing the proposed method. Indeed, these datasets have been designed for pure learning-to-rank approaches, where only the features are visible. For example, in order to avoid any privacy concern which could follow the public release of sensible data, LETOR does not provide any explicit information about the underlying queries and documents, which are used in the dataset. Furthermore, no additional information like the document content is provided. For all these reasons, it is very hard to introduce any relevant prior knowledge in this limited context.

Therefore, we decided to leverage the AOL dataset, released in 2005, to build a new learning-to-rank benchmark. The AOL dataset has been constructed from the logs of the AOL commercial search engine and it contains a sample of the search activity of 658000 anonymized US-based users over a three month period (March-May 2005), which has been estimated to consist of approximately 1.5% of the overall AOL users in the considered period. The dataset contains 4.8 million queries and 1.8 million URLs. Since we are aware of the privacy concerns of this dataset, the logs have been pruned to remove queries that have been issued less than 30 times and by fewer than four distinct users. This should remove personal queries and documents that could allow associating any anonymous user id to a real person. In this dataset we follow a similar approach to that proposed in [14] by assuming that the relevance of a document for a query is proportional to the number of times the users selected it (click-through-rate), in particular, the click-though-rate ranges in the $[0, 1]$ interval and it has been split into 7 portions of equal size. The first sub-interval is associated to a 0 relevance level (non relevant result), and the relevance level is constantly increased by 0.5 sub-interval by sub-interval up to a maximum relevance level equal to 3 (essential result). Therefore, the learning-to-rank problem consists in predicting the order of the documents in order to maximize the NDCG of the rank.

The dataset has been constructed by randomly selecting 10000 queries issued more than thirty times in the dataset. All the documents that have been selected for the query at least once by a user have been downloaded from the Internet. All the documents that were not available anymore at downloading time have been discarded, resulting into 140740 (query, document pairs). 40% of these queries are kept as candidates for inclusion in a training set. The validation and test sets have been created by randomly splitting the remaining queries into two groups containing 20% and 40% of the initial set, respectively. All the experiments presented in this section have been obtained as an average over 5-folds, obtained by subsampling the pool of reserved training data.

The downloaded HTML documents have been parsed and processed together with their associated query. The output of this process is a vectorial representation of each (query, document) pair composed by 140 features, 5 of which depending on the document only and 135 on the document and query. In particular, the entire document and 4 sections of the document are taken into account: title, body, url and anchor. For each portion, 27 features compute the match between the query and specific sub-portions of the document like the BM25, cosine similarity, etc. Most of these features have been implemented consistently to the LETOR dataset as reported in [1]. Furthermore, using a set of SVM text categorizers [3] built over the data available in the DMOZ taxonomy [3], the documents have

[2] http://research.microsoft.com/users/LETOR/

[3] http://www.dmoz.org

	RankSVM	RankNet	LambdaRank	SortNet	LambdaNeuralRank
NDCG@1	0.462	0.333	0.390	0.450	0.477
NDCG@2	0.563	0.400	0.542	0.568	0.579
NDCG@3	0.603	0.424	0.596	0.615	0.622
NDCG@4	0.628	0.438	0.624	0.639	0.644
NDCG@5	0.647	0.452	0.644	0.658	0.662
NDCG@6	0.663	0.464	0.660	0.672	0.676
NDCG@7	0.675	0.476	0.673	0.685	0.689
NDCG@8	0.686	0.486	0.683	0.696	0.699
NDCG@9	0.696	0.497	0.692	0.704	0.708
NDCG@10	0.705	0.50.7	0.700	0.712	0.716

Table 1: NDCG@n results on the AOL Web logs dataset for the proposed method and other state-of-the-art learning-to-rank approaches.

	Baseline (no constraints)	**Adult**	**Shopping**	**Health**	**Society**	**Sports**	**Recreation**	**Combined**
NDCG@1	0.375	0.425	0.427	0.423	0.398	0.436	0.423	0.435
NDCG@2	0.493	0.544	0.545	0.543	0.523	0.555	0.544	0.554
NDCG@3	0.547	0.600	0.596	0.591	0.573	0.602	0.584	0.598
NDCG@4	0.580	0.625	0.625	0.622	0.604	0.603	0.610	0.620
NDCG@5	0.603	0.645	0.645	0.642	0.624	0.650	0.628	0.640
NDCG@6	0.620	0.661	0.661	0.657	0.639	0.665	0.643	0.655
NDCG@7	0.635	0.672	0.673	0.670	0.653	0.678	0.657	0.668
NDCG@8	0.647	0.685	0.685	0.681	0.664	0.689	0.668	0.679
NDCG@9	0.657	0.694	0.694	0.690	0.674	0.698	0.678	0.689
NDCG@10	0.666	0.702	0.702	0.697	0.684	0.705	0.686	0.697
P@1	0.402	0.451	0.452	0.450	0.424	0.462	0.452	0.463
P@2	0.338	0.368	0.370	0.367	0.355	0.373	0.369	0.377
P@3	0.295	0.315	0.316	0.312	0.304	0.317	0.309	0.316
P@4	0.262	0.276	0.276	0.276	0.268	0.277	0.267	0.274
P@5	0.236	0.247	0.247	0.247	0.240	0.248	0.237	0.246
P@6	0.216	0.226	0.224	0.223	0.218	0.224	0.216	0.222
P@7	0.198	0.203	0.203	0.203	0.200	0.204	0.198	0.203
P@8	0.183	0.187	0.187	0.187	0.185	0.188	0.188	0.187
P@9	0.170	0.174	0.173	0.173	0.172	0.174	0.170	0.174
P@10	0.158	0.162	0.161	0.161	0.160	0.161	0.159	0.162
MAP	0.518	0.555	0.556	0.536	0.561	0.530	0.550	0.561

Table 2: NDCG@n, Precision@n and MAP results on the AOL Web logs dataset as a 5-fold average when using LambdaNeuralRank on 4 supervised and 4000 unsupervised queries per fold and a rule for each single category or all the categories combined. The *Baseline* column represents the results obtained by training without the rules.

been assigned a vector of eight scores in $[-1, 1]$ depending on how much their content belongs to the following categories: *Adult*, *Arts*, *Business*, *Shopping*, *Health*, *Society*, *Sports* and *Recreation*. These topicality scores has also been added to the vector of features. The queries, represented as their bag-of-words, have also been classified into the same categories (non-exclusively, meaning that a query can belong to multiple classes). Let $\mathcal{Q}$ be the set of all queries, we indicate with $\mathcal{Q}_c$ the subset of queries classified as belonging to category c. The ranking functions have been implemented by a 2-layer neural network with 40 hidden neurons with tahn activations. Cross-validation on the validation set has been performed to select the best performing neural network during the training process.

LambdaNeuralRank is a non-trivial extension of LambdaRank. Therefore, even if not the main focus of this paper, we start evaluating its performance before showing how it can be further improved using prior knowledge. In particular, table 1 reports the NDCG scores obtained on this dataset by various learning-to-rank approaches in the pairwise (RankSVM [9], RankNet [15] and SortNet [16]) and listwise category (LambdaRank [10]) against LambdaNeuralRank. This table shows that LambdaNeuralRank over-performs with a significant margin the other approaches for all values of NDCG@i. Therefore, LambdaNeuralRank provides a very solid learning-to-rank package, which is not trivial to improve.

The tested ranking rules where in the following form:

$$\forall q \in \mathcal{Q}_T \ \ \exists d \in \boldsymbol{d} \ \ r_k(q,d) \wedge T(d)$$

where $r_k(q,d)$ is a predicate in $\mathcal{R}$ returning true if d is one of the first k position in the rank for q and $T(d)$ is a predicates returning true if d is about topic T. This rule states that at least one result in the first k should be about a category if also the query is about that category.

In our experimental setting, $k = 3$ and, as said before, all the first level DMOZ categories have been considered.

Table 2 and 3 reports the NDCG and MAP scores on the test set as an average over 5-folds by randomly subsampling 4 and 40 queries as training set for each fold, respectively. The *Baseline* column reports the results provided by LambdaNeuralSort without integrating the logic knowledge. The other columns (excluding the last) show the results of the integration of the logic knowledge for a single class. The weight of the rule in each experiment has been determined via crossvalidation on the validation set. Finally, the *Combined* column reports the results provided by a classifier integrating all the logic knowledge at the same time. The weight of each rule λ_c^h in equation 1 has been set to a value proportional to the gain introduced by the rule on the validation set (more informative rules get higher weights). The benefit of the integration is very significant for both datasets, and it is very large on the experiment using few supervised queries. Please note that the additional logic helps generalization also for queries that do not belong to the category that is directly involved in the rule, for this reason it is possible to substantially increase the NDCG and P scores even when using singe rules. The *Combined* classifier is able

	Baseline (no constraints)	Adult	Shopping	Health	Society	Sports	Recreation	Combined
NDCG@1	0.419	0.426	0.444	0.435	0.416	0.414	0.436	0.454
NDCG@2	0.538	0.540	0.561	0.552	0.537	0.541	0.552	0.566
NDCG@3	0.586	0.589	0.608	0.600	0.585	0.587	0.589	0.608
NDCG@4	0.610	0.615	0.631	0.625	0.611	0.614	0.614	0.631
NDCG@5	0.632	0.635	0.651	0.646	0.629	0.635	0.633	0.649
NDCG@6	0.647	0.651	0.666	0.660	0.646	0.650	0.647	0.662
NDCG@7	0.660	0.664	0.678	0.672	0.659	0.663	0.660	0.676
NDCG@8	0.671	0.675	0.689	0.683	0.671	0.675	0.672	0.687
NDCG@9	0.681	0.684	0.698	0.692	0.680	0.685	0.683	0.696
NDCG@10	0.690	0.692	0.705	0.701	0.689	0.693	0.691	0.704
P@1	0.446	0.451	0.471	0.462	0.443	0.441	0.465	0.483
P@2	0.368	0.367	0.383	0.374	0.365	0.370	0.374	0.385
P@3	0.312	0.315	0.324	0.319	0.312	0.313	0.308	0.323
P@4	0.270	0.272	0.278	0.277	0.272	0.273	0.268	0.278
P@5	0.243	0.246	0.249	0.249	0.242	0.246	0.240	0.247
P@6	0.221	0.223	0.226	0.224	0.220	0.223	0.216	0.222
P@7	0.202	0.204	0.206	0.205	0.201	0.204	0.198	0.203
P@8	0.186	0.188	0.190	0.188	0.186	0.188	0.183	0.188
P@9	0.173	0.174	0.175	0.174	0.173	0.174	0.171	0.174
P@10	0.161	0.162	0.163	0.162	0.161	0.162	0.159	0.161
MAP	0.550	0.554	0.569	0.562	0.550	0.550	0.556	0.571

Table 3: NDCG@n, Precision@n and MAP results on the AOL Web logs dataset as a 5-fold average when training LambdaNeuralRank on 40 supervised and 4000 unsupervised queries per fold and a rule for each single category or all the categories combined. The *Baseline* column represents the results obtained by training without the rules.

to get use of the overall knowledge and it performs the best on most experiments.

6 Conclusions

This paper presents a novel approach to integrate prior knowledge in form of FOL clauses into learning-to-rank approaches. This methodology allows to get advantage of the usually extensive prior knowledge developed by human experts to build a given ranking function, without relying on a very large amount of training data to infer it. The experimental results show that this approach provides a large precision increase of the ranking function when little training data is available. This should allow to apply learning-to-rank approaches also in contexts where it is not economically possible to invest a large amount of time and money to label the training data.

7 Acknowledgments

This work has been partially founded by a research award from Google Inc. and by a grant from the MPS foundation. We thank Marco Gori, Marco Maggini, Nino Freno and Edmondo Trentin for the fruitful discussions that helped us to define this model.

REFERENCES

[1] T. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, "Letor: Benchmark dataset for research on learning to rank for information retrieval," in *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*, pp. 3–10, Citeseer, 2007.

[2] K. Crammer and Y. Singer, "Pranking with ranking," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 641–647, MIT Press, 2001.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96, ACM, 2005.

[4] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*, pp. 129–136, ACM, 2007.

[5] T. Qin, X. Zhang, M. Tsai, D. Wang, T. Liu, and H. Li, "Query-level loss functions for information retrieval," *Information Processing & Management*, vol. 44, no. 2, pp. 838–855, 2008.

[6] Y. Yue, T. Finley, F. Radlinski, and T. Joachims, "A support vector method for optimizing average precision," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 271–278, ACM, 2007.

[7] M. Volkovs and R. Zemel, "Boltzrank: learning to maximize expected ranking gain," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1089–1096, ACM, 2009.

[8] T. Qin, T. Liu, and H. Li, "A general approximation framework for direct optimization of information retrieval measures," *Information retrieval*, vol. 13, no. 4, pp. 375–397, 2010.

[9] Y. Cao, J. Xu, T. Liu, H. Li, Y. Huang, and H. Hon, "Adapting ranking svm to document retrieval," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 186–193, ACM, 2006.

[10] C. Burges, R. Ragno, and Q. Le, "Learning to rank with nonsmooth cost functions," *Advances in neural information processing systems*, vol. 19, p. 193, 2007.

[11] P. Donmez, K. Svore, and C. Burges, "On the local optimality of lambdarank," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 460–467, ACM, 2009.

[12] M. Diligenti, M. Gori, M. Maggini, and L. Rigutini, "Multitask Kernel-based Learning with Logic Constraints," in *Proceedings of the 19th European Conference on Artificial Intelligence*, pp. 433–438, IOS Press, 2010.

[13] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, no. 1, pp. 5–32, 1996.

[14] N. Craswell and M. Szummer, "Random walks on the click graph," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 239–246, ACM, 2007.

[15] P. Li, C. Burges, and Q. Wu, "Learning to rank using classification and gradient boosting," in *Proceedings of the International Conference on Advances in Neural Information Processing Systems (NIPS)*, Citeseer, 2007.

[16] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli, "SortNet: learning to rank by a neural-based sorting algorithm," in *In proceedings of the SIGIR 2008 Workshop on Learning to Rank for Information Retrieval (LR4IR)*, vol. 42, pp. 76–79, 2008.

Using SAT and SQL for Pattern Mining in Relational Databases

Emmanuel Coquery[1] and **Jean-Marc Petit**[1] and **Lakhdar Sais**[2]

Abstract. In this paper, we present an ongoing work bridging the gap between pattern mining, SQL and SAT for a particular class of patterns. We extend the work presented in [2] that proposes a logical query language for rule patterns satisfying Armstrong's axioms. Our contributions are the following: firstly, we allow a large part of the relational tuple calculus (SQL) to be used in the specification of queries. Secondly, we propose a boolean encoding of the query that can be used to compute answers even in the case of non Armstrong-compliant queries. Some experiments have been performed on top of Derby (embedded Java DBMS) and a modified version of MiniSat to show the feasibility of the approach.

1 INTRODUCTION

Declarative approaches for pattern mining attracted a growing attention in the recent years. On one hand, a way to increase declarativity is to devise high level query languages for data mining [10, 13, 14, 5, 8, 11, 2]. On the other hand, more declarativity often induce a greater expressivity, at the cost of a reduced efficiency. Constraint programming approaches for pattern mining, initiated in [15], were proposed to provide a good compromise between expressivity and efficiency.

In this paper, we propose to use SQL and SAT together for pattern mining. We extend the work presented in [2] that proposes a logical query language for rule patterns satisfying Armstrong's axioms. While such patterns can be enumerated with efficient algorithms they might be seen as too restrictive. The first contribution of this paper is to allow a large part of the relational tuple calculus (SQL) to be used in the specification of queries. The second one is the use of a SAT solver for enumerating patterns even in the case of non Armstrong-compliant queries.

The rest of this paper is organized as follows. Section 2 presents the $\mathcal{RLT}$ query language. Section 3 presents the basic principles of the query boolean encoding. Implementation principles are presented in section 4 together with a few optimizations, while section 5 provides some ways to reduce the number of answers. Finally some experimental results are presented in section 6 and we conclude in section 7.

2 THE $\mathcal{RLT}$ LANGUAGE

In this section, we introduce the syntax and semantics of the $\mathcal{RLT}$ language, which is based on a mixture of tuple relational calculus [1] and $\mathcal{RL}$ language [2].

2.1 Preliminaries

We introduce the following definitions and notations used in the $\mathcal{RLT}$ language:

- $\mathcal{U}$ is a set of *attributes*, noted $\bar{A}, \bar{B} \ldots$,
- $\mathcal{CST}$ is a set of constants,
- $\mathcal{CMP}$ is a set of binary[3] comparisons over $\mathcal{CST}$. It is assumed that if $\bar{c}$ and $\bar{c'}$ are two constants, and if $\square \in \mathcal{CMP}$, then $\bar{c} \square \bar{c'}$ is computable in constant time.
- a schema R is a finite, nonempty set of attributes from $\mathcal{U}$,
- a tuple $\bar{t}$ over a schema R is a total function from R to $\mathcal{CST}$,
- $\bar{t}[\bar{A}]$ denotes the value of $\bar{t}$ for attribute $\bar{A}$,
- a relation $\bar{r}$ over a schema R is a set of tuples over R.

- $s, t, u, s_1, \ldots$ are tuple variables,
- $A, B, C, A_1, B_1 \ldots$ are attribute variables, i.e. capital letters from the beginning of the alphabet,
- $X, Y, Z, X_1, Y_1 \ldots$ are schema variables, i.e. capital letters from the end of the alphabet,
- $r, r_1, r' \ldots$ are relation symbols.

To avoid ambiguity with variables, we shall use the following notations for attributes, set of attributes and tuples:

- $\bar{A}, \bar{B}, \bar{C}, \bar{A_1}, \bar{B_1} \ldots$ are single attributes,
- $\bar{X}, \bar{Y}, \bar{Z}, \bar{X_1}, \bar{Y_1} \ldots$ are set of attributes,
- $\bar{s}, \bar{t}, \bar{t_1}, \bar{t_2}, \ldots$ are tuples,
- $\bar{c}, \bar{c_1}, \bar{c'}$ are constants.

For any function $f : E \to E'$, we denote by $f[e := e']$, where $e \in E$ and $e' \in E'$, the function f' which maps e to e' and maps e_1 to $f(e_1)$ if $e \neq e_1$.

For any function $f : E \to E'$, and any subset $E_1 \subseteq E$, we denote by $f_{|E_1}$ the restriction $f' : E_1 \to E'$ of f to E_1, which maps $e_1 \in E_1$ to $f(e_1)$.

2.2 $\mathcal{RL}$ Formulas

This section recalls the syntax and semantics of the $\mathcal{RL}$ language [2]. In this paper, we restrict the use of tuple variable quantifiers, by removing them from definition 1 and reintroducing them in section 2.4. We also limit the use of attribute quantifiers to some form of restricted quantification.

[1] Université de Lyon, CNRS, France, email: firstname.lastname@liris.cnrs.fr
[2] Université d'Artois, CNRS, France, email: sais@cril.univ-artois.fr

[3] the *binary* restriction can be easily lifted, allowing arbitrary SQL boolean expressions

Let A, B be attribute variables, t, s tuple variables, $\bar{c}$ a constant, X a schema variable.

Definition 1 *The set of $\mathcal{RL}$-formulas, noted $\delta, \delta_1, \delta, \ldots$, is inductively defined as the smallest set verifying:*

- $t.A \ \Box \ \bar{c}$, $t.A \ \Box \ s.B$, $A = B$ *and* $A = \bar{A}$ *are* atomic *$\mathcal{RL}$-formulas*
- *If δ is a $\mathcal{RL}$-formula and A an attribute variable, $\forall A(X)(\delta)$ is a $\mathcal{RL}$-formula*
- *If δ is a $\mathcal{RL}$-formula and A an attribute variable, $\exists A(X)(\delta)$ is a $\mathcal{RL}$-formula*
- *If δ_1 and δ_2 are $\mathcal{RL}$-formulas, then $\neg\delta_1$ and $(\delta_1 \wedge \delta_2)$ are $\mathcal{RL}$-formulas*

Other logical connectors such as $\vee, \Rightarrow$ and abbreviations **true, false** are defined as usual.

Definition 2 *A* $\mathcal{RL}$-interpretation *is a quadruplet $(R, \Sigma, \sigma, \tau)$ where:*

- *$R \subseteq \mathcal{U}$ is a schema,*
- *Σ, the schema interpretation, is a function mapping each schema variable X to a subset of R,*
- *σ, the attribute interpretation, is a function mapping each attribute variable A to an attribute $\bar{A} \in R$,*
- *τ, the tuple interpretation, is a function mapping each tuple variable t to a tuple $\bar{t}$ over R.*

Definition 3 *Let δ be a $\mathcal{RL}$-formula. The* satisfaction *of δ with respect to a $\mathcal{RL}$* interpretation *$(R, \Sigma, \sigma, \tau)$, denoted by $(R, \Sigma, \sigma, \tau) \models \delta$, is defined inductively as follows:*

- $(R, \Sigma, \sigma, \tau) \models t.A \ \Box \ \bar{c}$ *if* $\tau(t)[\sigma(A)] \ \Box \ \bar{c}$
- $(R, \Sigma, \sigma, \tau) \models t.A \ \Box \ s.B$ *if* $\tau(t)[\sigma(A)] \ \Box \ \tau(t)[\sigma(B)]$
- $(R, \Sigma, \sigma, \tau) \models A = \bar{A}$ *if* $\sigma(A) = \bar{A}$
- $(R, \Sigma, \sigma, \tau) \models A = B$ *if* $\sigma(A) = \sigma(B)$
- $(R, \Sigma, \sigma, \tau) \models \forall A(X)(\delta)$ *if for all* $\bar{A} \in \Sigma(X)$, $(R, \Sigma, \sigma[A := \bar{A}], \tau) \models \delta$
- $(R, \Sigma, \sigma, \tau) \models \exists A(X)(\delta)$ *if for some* $\bar{A} \in \Sigma(X)$, $(R, \Sigma, \sigma[A := \bar{A}], \tau) \models \delta$
- $(R, \Sigma, \sigma, \tau) \models \neg\delta$ *if* $(R, \Sigma, \sigma, \tau) \not\models \delta$
- $(R, \Sigma, \sigma, \tau) \models (\delta_1 \wedge \delta_2)$ *if* $(R, \Sigma, \sigma, \tau) \models \delta_1$ *and* $(R, \Sigma, \sigma, \tau) \models \delta_2$

2.3 Relational Calculus

Here we recall some definitions of the tuple relational calculus [1] (abbreviated TRC in the following). Let $\bar{A}, \bar{B}$ be attributes, t, s be tuple variables, $\bar{c}$ be a constant and r be a relation symbol.

Definition 4 *The set of TRC-formulas, noted $\psi, \psi_1, \psi', \ldots$, is inductively defined as the smallest set verifying:*

- *$t.\bar{A} \ \Box \ c$ and $t.\bar{A} \ \Box \ t.\bar{B}$ are* atomic *TRC-formula*
- *$r(t)$ is an* atomic *TRC-formula*
- *if ψ is a TRC-formula, and t is a tuple variable then $\exists t(\psi)$ is a TRC-formula*
- *if ψ_1 and ψ_2 are TRC-formulas then $(\psi_1) \wedge (\psi_2)$ and $\neg(\psi_1)$ are TRC formulas*

Other logical connectors such as $\vee, \Rightarrow$, quantifier $\forall$ and abbreviations **true, false** are defined as usual.

Now we recall the logical semantics of TRC-formulas. For sake of simplicity, we assume that all relations and schemas are defined over the same schema R. This restriction can easily be lifted though.

Definition 5 *A TRC interpretation is a pair (d, τ) where:*

- *d is a function, the database, mapping each relation symbol r to a relation $\bar{r}$ over R,*
- *τ is a function, the tuple interpretation, mapping each tuple variable t to a tuple $\bar{t}$ over R.*

Definition 6 *Let ψ be a TRC-formula. The satisfaction of ψ with respect to a TRC interpretation (d, τ), denoted $(d, \tau) \models \psi$, is inductively defined as follows:*

- $(d, \tau) \models t.\bar{A} \ \Box \ \bar{c}$ *if* $\tau(t)[\bar{A}] = \bar{c}$
- $(d, \tau) \models t.\bar{A} \ \Box \ t.\bar{B}$ *if* $\tau(t)(\bar{A}) = \tau(t)(\bar{B})$
- $(d, \tau) \models r(t)$ *if* $\tau(t) \in d(r)$
- $(d, \tau) \models \neg(\psi)$ *if* $(d, \tau) \not\models \psi$
- $(d, \tau) \models \exists t(\psi)$ *if there exists a tuple $\bar{t}'$ over R such that* $(d, \tau[t := \bar{t}']) \models \psi$
- $(d, \tau) \models (\psi_1) \wedge (\psi_2)$ *if* $(d, \tau) \models \psi_1$ *and* $(d, \tau) \models \psi_2$

In the rest of the paper, we restrict ourselves to *authorized relational calculus* [1], a syntactical restriction of the relation calculus which garanties domain independence. That is, given a database d, the set of tuple interpretations τ such that $(d, \tau) \models \psi$ only depends on d and ψ.

Definition 7 *Given a TRC formula ψ, with $t_1, \ldots, t_k$ as free variables, the answer of ψ w.r.t. a database d, denoted by $ans(\psi, d)$, is defined as:*

$$\{\tau_{|\{t_1,\ldots,t_k\}} \mid (d, \tau) \models \psi\}$$

Note that for an authorized TRC formula ψ and a database d that associate only finite relations to relation symbols, the answer $ans(\psi, d)$ is finite.

2.4 $\mathcal{RLT}$ Queries

We introduce $\mathcal{RLT}$ queries, which constitute the $\mathcal{RLT}$-language.

Definition 8 *A $\mathcal{RLT}$ query is of the form:*

$$\{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k \ \psi \Rightarrow \delta\}$$

where

- *$X_1, \ldots, X_n$ are schema variables*
- *$t_1, \ldots, t_k$ are tuple variables over the same schema R*
- *ψ is an authorized TRC-formula, that has exactly $t_1, \ldots, t_k$ as free variables.*
- *δ is a $\mathcal{RL}$-formula in which:*
  - *the only (free) tuple variables are $t_1, \ldots, t_k$*
  - *the only (free) schema variables are $X_1, \ldots, X_n$*
  - *there is no free attribute variable.*

$\forall t_1 \ldots \forall t_k \ \psi \Rightarrow \delta$ is said to be the ($\mathcal{RLT}$) formula of the query.

In order to give an idea of how querying can be done using $\mathcal{RLT}$, let us consider the following query, which finds functional dependencies in a relation r over R:

$$Q_1 = \{\langle X, Y\rangle : R \mid \forall t \forall s (r(t) \wedge r(s)) \Rightarrow \\ (\forall A(X)(t.A = s.A)) \Rightarrow (\forall B(Y)(t.B = s.B))\}$$

Non Armstrong-compliant queries can also be expressed, such as:

$$Q_2 = \{\langle X, Y\rangle : R \mid \forall t \forall s (r(t) \wedge r(s)) \Rightarrow \\ (\exists A(X)(t.A = s.A)) \Rightarrow (\exists B(Y)(t.B = s.B))\}$$

or the following query for finding influence of genes

$$Q_3 = \{\langle X, Y\rangle : R \mid \forall t\ r(t) \wedge r(s) \wedge s.\bar{id} = t.\bar{id} + 1 \Rightarrow \\ (\forall A(X)(t.A > 0.6)) \Rightarrow (\forall B(Y)(t.B < 0.4))\}$$

assuming that $>$, $<$ and $\bar{c} = \bar{c'} + 1$ are comparisons in $\mathcal{CMP}$.

We now define the semantics of the language, first by defining the satisfaction of $\mathcal{RLT}$ query formulas, and then by defining the answers of a $\mathcal{RLT}$ query w.r.t. a given database.

Definition 9 *A $\mathcal{RLT}$ interpretation is a triple (d, R, Σ) where:*

- *$R \subseteq \mathcal{U}$ is a set of attributes;*
- *d is a function, the database, mapping each relation symbol r to a relation $\bar{r}$ over R;*
- *Σ, the schema interpretation, is a function mapping each schema variable X to a subset of R.*

Definition 10 *Let $\zeta = \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta$ be a $\mathcal{RLT}$-formula. ζ satisfies a $\mathcal{RLT}$ interpretation (d, R, Σ), denoted $(d, R, \Sigma) \models \zeta$, if for any tuple interpretation τ, and any attribute interpretation σ, if $(d, \tau) \models \psi$ then $(R, \Sigma, \sigma, \tau) \models \psi$.*

Taking any tuple interpretation corresponds to the use of $\forall$ quantifiers in $\mathcal{RLT}$ formulas. On the other hand, attribute interpretation are not important since δ does not contain free variables.

Definition 11 *Given a database d and a $\mathcal{RLT}$ query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$, the answer of Q in d, denoted by $ans(Q, d)$ is defined as:*

$ans(Q, d) = \{\langle \Sigma(X_1), \ldots, \Sigma(X_n)\rangle \mid (d, R, \Sigma)$ is a $\mathcal{RLT}$ interpretation and $(d, R, \Sigma) \models \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$

3 BOOLEAN ENCODING

In order to find answers of a query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$ w.r.t. a database d, we propose an encoding of the query and the database into a boolean formula. More precisely, for each interesting tuple interpretation τ, we generate a boolean formula representing the truth value of δ w.r.t. Σ. A tuple interpretation is considered to be interesting if, together with d, it satisfies ψ. The boolean formula for computing $ans(Q, d)$ is then the conjunction of the formulas for each interesting tuple interpretation.

3.1 Translation To Boolean Formula

Domain The domain, that is the boolean variables encoding an answer in $ans(Q, d)$, is defined straightforwardly as follows: for each schema variable $X \in \{X_1, \ldots, X_n\}$ and each attribute $\bar{A} \in R$, the boolean variable $p^X_{\bar{A}}$ is true whenever $\bar{A} \in \Sigma(X)$.

The following definition explains how the boolean formula is built from the $\mathcal{RL}$ formula. This definition relies on an attribute interpretation. However, this interpretation has no influence on top-level $\mathcal{RL}$ formulas, as they have no free attribute variable.

Definition 12 *Given a $\mathcal{RL}$ formula δ, a tuple interpretation τ, an attribute interpretation σ and a set of attributes R, the boolean encoding of δ, denoted by $enc(\delta, \tau, \sigma, R)$, is inductively defined as:*

- *$enc(t.A = \bar{c}, \tau, \sigma, R) =$ **true** if $\tau(t)[\sigma(A)] = \bar{c}$, **false** otherwise*
- *$enc(t.A = s.B, \tau, \sigma, R) =$ **true** if $\tau(t)[\sigma(A)] = \tau(s)[\sigma(B)]$, **false** otherwise*
- *$enc(A = \bar{A}, \tau, \sigma, R) =$ **true** if $\sigma(A) = \bar{A}$*
- *$enc(A = B, \tau, \sigma, R) =$ **true**) if $\sigma(A) = \sigma(B)$, **false** otherwise*
- *$enc(\neg\delta, \tau, \sigma, R) = \neg enc(\delta, \tau, \sigma, R)$*
- *$enc(\delta_1 \wedge \delta_2, \tau, \sigma, R) = enc(\delta_1, \tau, \sigma, R) \wedge enc(\delta_2, \tau, \sigma, R)$*
- *$enc(\forall A(X)\delta, \tau, \sigma, R) =$ $\bigwedge_{\bar{A} \in R}(\ p^X_{\bar{A}} \Rightarrow enc(\delta, \tau, \sigma[A := \bar{A}], R)\ )$*
- *$enc(\exists A(X)\delta, \tau, \sigma, R) =$ $\bigvee_{\bar{A} \in R}(\ p^X_{\bar{A}} \wedge enc(\delta, \tau, \sigma[A := \bar{A}], R)\ )$*

Definition 13 *A* boolean interpretation *I is a function from boolean variables to* $\{$**true**, **false**$\}$. *It* satisfies *a boolean formula γ, denoted by $I \models \gamma$, if the formula obtained by replacing each variable p by $I(p)$ is equivalent to* **true** *in the boolean algebra.*

Property 1 *Let Σ be a schema interpretation, δ a $\mathcal{RL}$ formula, τ a tuple interpretation, σ an attribute interpretation and R a set of attributes. Let I_Σ be a boolean interpretation such that for any schema variable X and any attribute $\bar{A}$, $I_\Sigma(p^X_{\bar{A}}) =$ **true** if and only if $\bar{A} \in \Sigma(X)$.*

Then $(R, \Sigma, \sigma, \tau) \models \delta$ if and only if $I_\Sigma \models enc(\delta, \tau, \sigma, R)$.

Proof By definitions 3 and 12.

Definition 14 *The boolean encoding of a query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$ w.r.t. a database d, denoted by $enc(Q, d)$ is defined as:*

$$\bigwedge_{\tau \in ans(\psi, d)} enc(\delta, \tau, \sigma, R)$$

where σ is any attribute interpretation[4].

Property 2 *Let $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$ be a $\mathcal{RLT}$ query, d a database and Σ a schema interpretation. Let I_Σ be a boolean interpretation such that for any schema variable X and any attribute $\bar{A}$, $I_\Sigma(p^X_{\bar{A}} =$ **true**) if and only if $\bar{A} \in \Sigma(X)$.*

$\langle \Sigma(X_1), \ldots, \Sigma(X_n)\rangle \in ans(Q, d)$ if and only if $I_\Sigma \models enc(Q, d)$.

[4] σ is not actually used in the encoding since δ is closed w.r.t. attribute variables

Proof By definitions 11, 7 and 14, by property 1 and by remarking that if $\tau_{|\{t_1,\ldots,t_k\}} = \tau'_{|\{t_1,\ldots,t_k\}}$, then $enc(\delta, \tau, \sigma, R) = enc(\delta, \tau', \sigma, R)$.

3.2 Theoretical Complexity

The cost of evaluating a $\mathcal{RLT}$ query using a boolean formula can be evaluated by the size of the formula and its number of boolean variables. Except for quantifiers, each construction of $\mathcal{RL}$ formula generate only a constant amount of additional symbols in the encoding. For each use of a quantifier $\Box A(X)\delta$, if the size of $enc(\delta, \tau, \sigma, R)$ is n, then the size of $enc(\Box A(X)\delta, \tau, \sigma, R)$ is in $O(|R| \times n)$.

Let us consider a $\mathcal{RLT}$ query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$. Let $n_{\forall\exists}$ be the maximal number of quantifiers on a branch of the abstract syntax tree of δ. Then, an upper bound on the size of $enc(Q, d)$ is $O(|ans(\psi, d)| \times |\delta| \times |R|^{n_{\forall\exists}})$.

Let $time_{\psi,d}$ be the time required to evaluate $ans(\psi, d)$. Then an upper bound on the time complexity of the evaluation of a $\mathcal{RLT}$ query is $time_{\psi,d} + O(|ans(\psi, d)| \times |\delta| \times |R|^{n_{\forall\exists}} \times 2^{n \times |R|})$.

4 IMPLEMENTATION AND OPTIMIZATION

In this section we present the principles used in the implementation of $\mathcal{RLT}$, as well as a few optimizations that help to drastically reduce the size of generated formulas. In this section, we assume given both $\mathcal{RLT}$ query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$ and database d. Figure 1 presents the different steps used in the computation of answers.

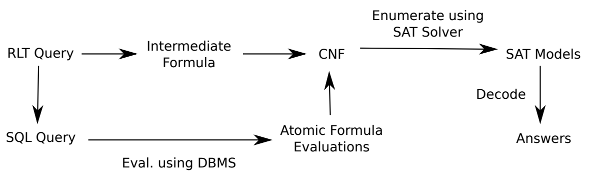


Figure 1. Architecture

4.1 Naive translation

The first step in generating $enc(Q, d)$ is to evaluate answers $ans(\psi, d)$. Since ψ is an authorized TRC formula, it can evaluated using a SQL engine, at the cost of an attribute renaming to avoid attribute name clashes between tuples. We will see in section 4.3 that in the final implementation this problem disappears. Using SQL allows to easily extend the comparison predicates and expressions used in the TRC formula ψ. Then for each tuple combination τ we generate $enc(\delta, \tau, \sigma, R)$, with σ being uninitialized[5].

4.2 Getting Answers From Boolean Formula

Because of property 2, the answers $ans(Q, d)$ can be obtained by the boolean interpretations satisfying the formula $enc(Q, d)$. We use a modified SAT-solver [6] based on Minisat [7]. Using a SAT solver requires the formula to be translated into conjunctive normal form (CNF). For this we use a linear translation based on [16]. This translation propagates constants through standard logical equivalences such as $\eta_1 \wedge \mathbf{true} \equiv \eta_1$ and $\eta_1 \wedge \mathbf{false} \equiv \mathbf{false}$. The translation into CNF also introduce new variables. However, the value of these new variables can be deduced from the values of the variables in the original formula. This information can be transmitted to the enumerating SAT solver, allowing it to branch only on variables of the original formula. Moreover the use of modern SAT solvers allows to benefit from efficient propagation techniques, learning [12, 17] and dynamic search heuristics [17, 4]. For an extensive overview of current techniques to solve SAT, the reader is referred to [3].

4.3 Caching Subformulas

By looking at definition 12, one can see that the generated formula for each tuple interpretation depend on the structure of δ and on the evaluation of the atomic sub formulas of δ for (all) the possible attribute interpretations σ. That is, the actual value of tuples is not important, only the value they give to atomic formulas in δ matters.

For each atomic formula δ_1 in δ, we introduce a series of boolean variables $q^{\delta_1}_{\sigma}$ for all possible attribute interpretation $\sigma : \mathcal{A} \to R$ where $\mathcal{A}$ is the set of attribute variables appearing in δ_1.

Definition 15 *The* intermediate encoding $enc'(\delta, \sigma, R)$ *is inductively defined as follows:*

- $enc'(t.A = \bar{c}, \sigma, R) = q^{t.A=\bar{c}}_{\sigma_{|\{A\}}}$
- $enc'(t.A = s.B, \sigma, R) = q^{t.A=s.B}_{\sigma_{|\{A,B\}}}$
- $enc'(A = \bar{A}, \sigma, R) = \mathbf{true}$ *if* $\sigma(A) = \bar{A}$
- $enc'(A = B, \sigma, R) = \mathbf{true})$ *if* $\sigma(A) = \sigma(B)$*,* **false** *otherwise*
- $enc'(\neg\delta, \sigma, R) = \neg enc'(\delta, \sigma, R)$
- $enc'(\delta_1 \wedge \delta_2, \sigma, R) = enc'(\delta_1, \sigma, R) \wedge enc'(\delta_2, \sigma, R)$
- $enc'(\forall A(X)\delta, \sigma, R) = \bigwedge_{\bar{A}\in R}(\ p^X_{\bar{A}} \Rightarrow enc'(\delta, \sigma[A := \bar{A}], R)\ )$
- $enc'(\exists A(X)\delta, \sigma, R) = \bigvee_{\bar{A}\in R}(\ p^X_{\bar{A}} \wedge enc'(\delta, \sigma[A := \bar{A}], R)\ )$

This definition is similar to definition 12, except for atomic formulas where tuple variables appear. By construction, for a given tuple interpretation τ, $enc(\delta, \tau, \sigma, R)$ can be obtained by replacing each variable $q^{\delta_1}_{\sigma_1}$ in $enc'(\delta, \sigma, R)$ by $enc(\delta_1, \tau, \sigma_1, R)$.

This suggests a change in the generation of $enc(Q, d)$: the SQL engine can be used to evaluate the value of atomic formulas in δ w.r.t. all relevant attribute interpretations σ, that is w.r.t. all combination of values for attribute variables appearing in the sub formula. The encoding of the $\mathcal{RL}$ formula for this tuple combination is then obtained by propagating these boolean values in the intermediate encoding.

The benefits of this change are twofold. Firstly, the comparison operator and expressions used in atomic $\mathcal{RL}$ formulas can easily be extended to any operator supported by the SQL engine. Secondly, and more importantly, given two tuple interpretations τ_1 and τ_2, if the evaluation of all atomic formulas w.r.t. all attribute interpretation are the same for τ_1 and τ_2, then $enc(\delta, \tau_1, \sigma, R) = enc(\delta, \tau_2, \sigma, R)$. Since the occurrence of

[5] In fact, we just initialize the data structure for representing the function

these two formulas are used in same conjunction, one of them is useless and can be discarded. This discarding behavior can be obtained, either by the use of the DISTINCT keyword in the SQL query, or more efficiently, by the use of a prefix tree to store and search for atomic formula evaluations. This optimization can be essential for the diminution of the size of the generated formula as shown in section 6.

4.4 Attribute Variable Combinatorics

Another way to reduce the size of $enc(Q, d)$ is to try to reduce the number of nested attribute quantifiers in δ. We assume, without loss of generality, that each attribute variable appears exactly in one quantifier in δ. The attribute quantifiers can be "pushed down" towards atomic formulas in δ, by using the following standard logical equivalences:

- $\forall A(X)(\delta_1 \wedge \delta_2) \equiv (\forall A(X)\delta_1) \wedge \delta_2$ if A does not appear in δ_2
- $\exists A(X)\neg\delta_1 \equiv \neg\forall A(X)\delta_1$
- $\forall A(X)\forall B(Y)\delta_1 \equiv \forall B(Y)\forall A(X)\delta_1$

For example, using these equivalences $\exists A(X)\forall B(Y)(t.A = 1 \Rightarrow t.B = 1) \equiv (\forall A(X)t.A = 1) \Rightarrow (\forall B(Y)t.B = 1)$. The size of the generated formula in the second case is $O(|R|)$ smaller than the one generated in the first case.

The use of $\wedge$ commutativity and associativity may allow for more optimizations such as $\forall A(X)\forall B(Y)(\delta_1 \wedge (\delta_2 \wedge \delta_3)) \equiv \forall B(Y)(\delta_2) \wedge \forall A(X)(\delta_1 \wedge \delta_3)$ if A does not appear in δ_2 and B does not appear in δ_1 nor δ_3. From this point of view, this kind of optimization can be brought near rule based optimization in relational queries [1].

5 REDUCING RESULT SIZE

As the search space size is $2^{n\times|R|}$, it is interesting to reduce the number of results. For example, it is usual when mining functional dependancies to output a minimal base of rules from which all rules can be inferred using Armstrong's axioms [9]. However, since our language is not supposed to be Armstrong-compliant [2], we express a wider class of queries without knowing a priori whether or not a given property is true (e.g. transitivity or reflexivity).Thus a canonical condensed representation of rules may not exist. Nevertheless, we provide means to end-users to reduce the number of results, while keeping interesting information. These means come in two flavors: firstly constraining the resulting sets of attributes, and secondly output only minimal sets (or maximal) w.r.t. set inclusion for some schema variables.

5.1 Constraining schema variables

The following examples illustrate how $\mathcal{RL}$ formulas can be used to constrain schema variables. Assume one wants to constraint two schema variables X and Y, such that $\Sigma(X) \cap \Sigma(Y) = \emptyset$. This constraint can be expressed by $\forall A(X)\forall B(Y)\ \neg A = B$. The formula $\exists A(X)$ **true** imposes that $\Sigma(X)$ contains at least one attribute, while the formula $\forall A(X)\forall B(X)\ A = B$ imposes that $\Sigma(X)$ contains at most one attribute.

One can remark that $(R, \Sigma, \sigma, \tau) \models \exists A(X)X = \bar{A}$ if and only if $\bar{A} \in \Sigma(X)$. Therefore $enc(\exists A(X)X = \bar{A}, \tau, \sigma, R) \equiv p^X_{\bar{A}}$[6]. This allows constraining schema variables by using any boolean formula on the variables $p^X_{\bar{A}}$ through the $\mathcal{RL}$ formula of an $\mathcal{RLT}$ query. A consequence of this remark is that given d and Q, the problem of determining whether $ans(Q, d) \neq \emptyset$ is NP-Hard.

5.2 Minimizing/Maximizing Schema Variables

Another way to reduce the number of results is to minimize or maximize schema interpretation values for some variables.

Definition 16 *A schema interpretation Σ is said to be* minimal *(resp.* maximal*) w.r.t. a schema variable X, a database d and $\mathcal{RLT}$ query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \zeta\}$, if there is no R_X such that $R_X \subset \Sigma(X)$ (resp. $R_X \supset \Sigma(X)$) and $(d, R, \Sigma[X := R_X]) \models \zeta$.*

Σ is said to be locally *minimal (resp. maximal) w.r.t. X, d and Q if there is no R_X such that $R_X \subset \Sigma(X)$ with $|R_X| = |\Sigma(X)| - 1$ (resp. $R_X \supset \Sigma(X)$ with $|R_X| = |\Sigma(X)| + 1$) and $(d, R, \Sigma[X := R_X]) \models \zeta$.*

Q is said to be monotone (resp. antimonotone*) w.r.t. X if for all d, Σ and R_X such that $\Sigma(X) \subset R_X \subseteq R$ (resp. $R_X \subset \Sigma(X)$), if $(d, R, \Sigma) \models \zeta$ then $(d, R, \Sigma[X := R_X]) \models \zeta$.*

It is clear that if Q is monotone (resp. antimonotone) w.r.t. X, then if Σ is locally minimal (resp. maximal) w.r.t. X, d and Q then it is maximal (resp. minimal) w.r.t. X, d and Q. We propose the following boolean encoding of the locally minimal/maximal constraint on a schema variable X, a database d and a $\mathcal{RLT}$ query $Q = \{\langle X_1, \ldots, X_n\rangle : R \mid \forall t_1 \ldots \forall t_k\ \psi \Rightarrow \delta\}$. Given a boolean formula γ, we denote by $\gamma[\gamma'/p]$ the formula obtained by replacing each occurrence of p in γ by γ'.

- $enc_{min}(X, d, Q) = \bigwedge_{\bar{A}\in R} p^X_{\bar{A}} \Rightarrow \neg(enc(Q, d)[\mathbf{false}/p^X_{\bar{A}}])$
- $enc_{max}(X, d, Q) = \bigwedge_{\bar{A}\in R} \neg p^X_{\bar{A}} \Rightarrow \neg(enc(Q, d)[\mathbf{true}/p^X_{\bar{A}}])$

The size of this constraint's boolean encoding is $|R|$ times the size of the original query's boolean encoding.

6 PRELIMINARY EXPERIMENTS

The CNF generator has been coded in Java, while the modified MiniSat solver in C++. We have used an embedded DBMS (Derby), since it allows to include the execution of SQL statements in CPU time results. The experiments were conducted on a 2GHz dual core Athlon processor with 3GB of RAM, running Linux.

This section presents a few experimental results on the following $\mathcal{RLT}$ query:

$$\begin{aligned}\{\langle X, Y\rangle : R \mid\ & \forall t_1\forall t_2\ r(t_1) \wedge r(t_2) \Rightarrow\\ & ((\forall A(X)t_1.A = t_2.A) \Rightarrow (\forall B(Y)t_1.B = t_2.B))\\ & \wedge(\forall A(X)\forall B(Y)\ \neg A = B)\\ & \wedge(\exists B(Y)\ \mathbf{true}) \wedge (\forall B_1(Y)\forall B_2(Y)\ B_1 = B_2)\}\end{aligned}$$

This query finds functional dependancies $X \rightarrow Y$ in r, X and Y having an empty intersection and Y being a singleton. Moreover X was minimized.

The relation initially contains 2013 tuples and 27 attributes. Figure 2 shows evolution of CPU time w.r.t. the number m

[6] This simplification is automatically performed through the propagation of **true** and **false** in the generated boolean formula.

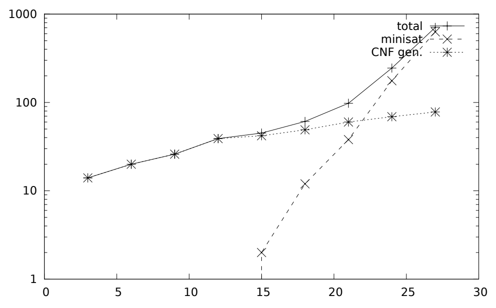


Figure 2. CPU time (in sec.) w.r.t. $|R|$

of attributes in R, *i.e.* only m attributes were kept in the relation r. As expected, the CPU time increases exponentially w.r.t. the number of attributes, as it increases both the search space and the size of the boolean formula. Figure 3 shows the

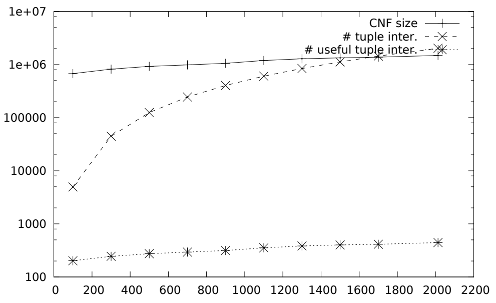


Figure 3. cache influence w.r.t. # tuples in r

size of the generated CNF (in number of variable occurrences) and the number of attribute interpretations w.r.t. the number of tuples, the query being run on the 27 attributes of the relation. As expected, the number of interpretations grows quadratically as there are two uncorrelated tuple variables in the query. One can remark that the size of the CNF, increases slowly. This is due to the low number of useful additional tuple interpretations. Indeed the size of the generated CNF increases proportionally to the number of useful tuple interpretations. This shows the efficiency of the cache optimization on boolean formula generation.

7 CONCLUSION

We presented an ongoing work on the query language $\mathcal{RLT}$ for pattern mining. Namely, we presented the semantics of the language, as well as a translation of queries and data into a boolean formula. Implementation techniques used for implementing a query engine were presented and some experimental results show the feasibility of the approach.

Several issues remain to be explored. One the theoretical side, it would be interesting to characterize the complexity of answering $\mathcal{RLT}$ queries (e.g. the complexity of determining the emptiness of a query). One the practical side, performances of the query engine and query optimization techniques have to be investigated through a comprehensive set of databases.The performance of the current implementation could be improved either through high level optimization in order to generate better, more easy to solve, formulas, or through the elaboration of (SAT) engines dedicated to the enumeration of models of boolean formulas. An other direction of improvement is to enrich the language, for example with counting statements to be able to take into account the well-known frequency constraint in data mining. This could be treated using pseudo-boolean constraints such as in [15].

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison Wesley, 1995.

[2] Marie Agier, Christine Froidevaux, Jean-Marc Petit, Yoan Renaud, and Jef Wijsen, 'On Armstrong-compliant Logical Query Languages', in *4th International Workshop on Logic in Databases (LID 2011)*, pp. 33–40. ACM, (March 2011).

[3] *Handbook of Satisfiability*, eds., A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009.

[4] Armin Biere, 'Adaptive restart strategies for conflict driven SAT solvers', in *Theory and Applications of Satisfiability Testing*, pp. 28–33, (2008).

[5] Toon Calders and Jef Wijsen, 'On monotone data mining languages', in *8th International Workshop on Database Programming Languages*, (2001).

[6] Emmanuel Coquery, Saïd Jabbour, and Lakhdar Sais, 'A constraint programming approach for enumerating motifs in a sequence', in *Workshop on Declarative Pattern Mining, ICDM Workshops*, pp. 1091–1097, (2011).

[7] Niklas Eén and Niklas Sörensson. Minisat. `http://minisat.se/`.

[8] Fosca Giannotti, Giuseppe Manco, and Franco Turini, 'Towards a logic query language for data mining', in *Database Support for Data Mining Applications, LNCS 2682*, pp. 76–94, (2004).

[9] G. Gottlob and L. Libkin, 'Investigations on Armstrong relations, dependency inference, and excluded functional dependencies', *Acta Cybernetica*, **9**(4), 385–402, (1990).

[10] Tomasz Imielinski and Heikki Mannila, 'A database perspective on knowledge discovery', *Commun. ACM*, **39**(11), 58–64, (1996).

[11] Hong-Cheu Liu, Aditya Ghose, and John Zeleznikow, 'Towards an algebraic framework for querying inductive databases', in *DASFAA (2)*, pp. 306–312, (2010).

[12] Joao P. Marques-Silva and Karem A. Sakallah, 'GRASP - A New Search Algorithm for Satisfiability', in *Proceedings of International Conference on Computer-Aided Design*, pp. 220–227, (1996).

[13] Rosa Meo, Giuseppe Psaila, and Stefano Ceri, 'An extension to sql for mining association rules', *Data Min. Knowl. Discov.*, **2**(2), 195–224, (1998).

[14] Amir Netz, Surajit Chaudhuri, Jeff Bernhardt, and Usama M. Fayyad, 'Integration of data mining with database technology', in *VLDB*, pp. 719–722, (2000).

[15] Luc De Raedt, Tias Guns, and Siegfried Nijssen, 'Constraint programming for itemset mining', in *KDD*, pp. 204–212, (2008).

[16] G.S. Tseitin, 'On the complexity of derivations in the propositional calculus', in *Structures in Constructives Mathematics and Mathematical Logic, Part II*, ed., H.A.O. Slesenko, pp. 115–125, (1968).

[17] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik, 'Efficient conflict driven learning in Boolean satisfiability solver', in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 279–285, (2001).

XQuake as a Constraint-Based Mining Language

Valerio Grossi and **Andrea Romei** [1]

Abstract. XQuake is a language for data mining inspired by the inductive databases theory. This work extends XQuake with the definition of domain-specific constraints. An ontology is used to describe the domain knowledge. We give the main idea of the work-in-progress discussing its possibilities and advantages.

1 INTRODUCTION AND MOTIVATION

The use of constraints in a mining application has rapidly become a challenging topic for research community. The aim is to find a unified approach for the definition of a *constraint-based mining language*. The use of constraints helps the analyst to model mining problems by specifying desirable properties of the mined patterns. This feature involves several aspects. In fact, even when the set of *domain-specific constraints* is known, it is challenging to provide both a query language for formalizing them and a system that can process their properties in an efficient way. For example, the extraction of association rules implies the discovery of a large quantity of useless rules. An approach that permits to extract rules by specifying properties based on the analyst needs is required.

Our goal is the definition of a constraint-based language that enables the specification of domain-specific constraints with the aid of an ontology. Specifically, this paper proposes an extension of XQuake [5, 6], a language and system for supporting complex mining tasks out of XML data. Surprisingly, no previous work has addressed the use of both ontology-based mining constraints and, at the same time, a coherent and uniform framework for representing data, mining models, and the KDD process. Since XQuake already supports the second aspect, we concentrate on its extension to express domain-dependent constraints.

2 XQUAKE AT A GLANCE

XQuake is a language and system for programming data mining processes over native XML databases, in the spirit of inductive databases [4]. While we refer the reader to [5] for a detailed description, here we summarize its main features as follows: *(i)* it satisfies the closure principle, since both data and mining models are represented in XML; *(ii)* it represents the KDD process in a declarative way, by means of an XQuery [9] program extended with mining primitives; *(iii)* it permits to evaluate constraints via XQuery predicates, with the aid of a built-in library; *(iv)* the system architecture is conceived to automatically capture the proprieties of such constraints (e.g. monotonicity and anti-monotonicity), to be used directly during the extraction of the mining model.

[1] Department of Computer Science, University of Pisa, Italy. Email: {vgrossi,romei}@di.unipi.it

The question now is, why the choice of a mining language out of XML data? We provide three answers. *(i)* The amount of information XML-coded is steadily growing. *(ii)* XQuake takes advantage of the XML philosophy also for representing the results of the mining process, according to the PMML standard [7]. *(iii)* Ontologies represented as OWL [8] are de facto XML documents, and they can be used to describe a domain knowledge. This latter point is exactly what we introduce in the next section.

3 A SIMPLE APPLICATION SCENARIO

We propose a scenario involving a simple Market Basket Analysis application inspired by our previous work [2]. The study of constrained association rule mining and constraint programming is well-known in literature (e.g. see [3]). Here, we introduce a mining language for defining, in an uniform approach, both mining tasks and domain-specific constraints on the extracted knowledge.

Application scenario. Let us suppose that a domain expert investigates for a future promotional campaign during the holidays. The goal is to study the relation between the most frequent drinks promoted in Easter, and the most frequent cakes promoted in Christmas in the past.

Input data. Data of a national supermarket has been translated from a relational format (see [2]) to an XML format, stored in the native XML database of XQuake. An XML fragment of the `MBA.xml` document is reported below.

```
<MBA>
  <store @id="id00193">
    <purchase @date="05/01/2012">
      <item @qty="2" @price="3.5">vodka lemon</item>...
    </purchase>...
  </store>...
</MBA>
```

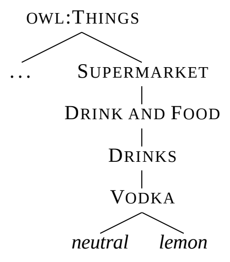
Here, each `<store>` tag represents a sequence of `<purchase>` elements, each encoding our transaction. The attribute `@date` denotes the date of that purchase. Each transaction is made up of a not empty set of purchased items, encoded in the `<item>` XML element. Two XML attributes denote the quantity and the price of that item. The resulting XML database contains 775,000 transactions and about 30,000 distinct items.

Domain knowledge. We enrich our data with the domain knowledge represented as an OWL document containing a description of each item and their hierarchical organization. A fragment of the `items.owl` ontology is depicted in Fig. 1.

Data constraints. Input data is filtered by considering *(i)* only purchases made between the 1^{st} December and the 1^{st} May and, *(ii)* for each transaction, only the items having a total price (i.e. `@qty` * `@price`) greater than 5 Euros.

Simple knowledge constraints. We aim at extracting association rules having exactly one item in the consequent, a minimum support of 0.5 and a minimum confidence of 0.7.

Advanced domain-specific constraints. Our purpose is to extract



attribute	value
hasColour	*neutral*
hasAlcoholic	*high*
hasFlavour	*lemon*
hasBrand	*Best*
hasSize	*70 cl*
hasPrice	*EUR 7.56*
hasPromotion	*Christmas*

Figure 1. A fragment of the hierarchical structure of `items.owl` (left). Data properties for 'Vodka Lemon' at the lower level (right).

```
(: Transactions, items and data constraints specif. :)
1.  for data $trans in doc("MBA")/store/purchase
       [@date > "01/01/2012" and @date < "01/05/2012"]
2.  let group $item := $trans/item[@price * @qty > 5.0]

(: Domain knowledge specification :)
3.  let supplementary $hasRec :=
      (<drink>{local:hasRec("Drink","Easter")}</drink>,
       <cake>{local:hasRec("Cake","Christmas")}</cake>)

(: Output constraints specification :)
4.  having (some $v in mfn:supplementary-body()
             satisfies $v/drink) and
5.    count(mfn:supplementary-head()) = 1 and
6.    mfn:supplementary-head[1]/cake and
7.    mfn:support() > 0.50 and mfn:confidence() > 0.70

8. return pmml
```

Figure 2. A possible implementation of the MBA scenario with XQuake.

association rules of the following form:

$$(i_1 \in \text{EasterDrink}) \text{ and } (i_2 \in \text{AnyItem}) \text{ and ... and } (i_n \in \text{AnyItem}) \Rightarrow (i_{n+1} \in \text{ChristmasCake}) \text{ [supp] [conf]}$$

where `EasterDrink` (resp. `ChristmasCake`) is the class of items that are drinks (resp. cakes) having an Easter (resp. Christmas) promotion, and `AnyItem` is the entire set of distinct items.

Our proposed solution. Among the plethora of languages for querying RDF and OWL documents, we integrate in XQuake an adaptation of [1], where, XQuery is employed for querying and reasoning with OWL and RDF ontologies. This solution appears to be adequate to our purposes for two main reasons. First, since XQuake extends XQuery with mining primitives, in turn, the implementation of an extension of XQuery for querying OWL documents is quite intuitive to be used by the user. Second, this permits to maintain the principle of closure that is at the basis of the inductive databases theory.

Fig. 2 provides a snapshot of the XQuake implementation, for further details see [5, 6] and [1]. The query is composed by four parts. First, the set of transactions (i.e. the `<purchase>` elements) is specified by the `for data` clause (row 1). Items of each transaction (i.e. the elements `<item>`) are defined in the `let group` clause (row 2).

Second, the `let supplementary` clause is evaluated for each frequent item (row 3). Here, a pair of XML elements are constructed by using the user-defined function reported below[2]:

```
declare function local:hasRec($class, $prop)
as xs:boolean {
  let $owl := owldoc("items.owl")
  return sw:hasSuperclass($owl, mfn:item(), $class)
         and sw:hasProperty($owl, mfn:item(),
                        $prop, "hasRecurrency")]
};
```

Third, the `having` clause operates on the output result (rows 4-7). It contains an XQuery predicate that is evaluated for each mined association rule. Basically, it evaluates the required constraints on the domain knowledge defined at row 3[3].

Finally, the `return` clause (row 8) is evaluated once, to return a PMML document containing the association rules satisfying the constraints.

Flexibility. It is important to note that our language is flexible both as far as a *modification of the domain knowledge* (since we use special constructs to traverse the ontology) and as far as the *introduction of different kinds of constraints* (since we use XQuery predicates for expressing them).

[2] If the current frequent item i (obtained through the built-in mining function `mfn:item()`) belongs to the superclass c in the ontology d, the built-in function `sw:hasSuperclass(d,i,c)` returns true. The build-in function `sw:hasProperty(d,i,p,v)` returns true if, for the item i, the data property p has value v in d

[3] The `mfn:supplementary-body()` (resp. `mfn:supplementary-head()`) built-in function returns the domain knowledge associated to the items of the body (resp. head) of the rule.

4 CONCLUSION

This work focuses on a general solution for XML data mining, and more generally, for a data mining query language. The solution proposed, even if at a preliminary stage, gives an additional improvement, in terms of the definition of complex domain-specific constraints. Our main advantages consist in a framework in which one can define, in an **expressive**, **declarative** and **uniform** way, both the KDD process and the mining constraints on the extracted knowledge. While the former aspect has been discussed in [6], this paper investigated the latter aspect. Basically, constraints are expressed through XQuery expressions supported by a built-in library. The proposed study can be refined in several directions. Here, we mention only the exploitation of domain-specific constraints for other kinds of models, such as sequential patterns and clusters.

ACKNOWLEDGMENTS

This work was supported by the EU FET-Open project "ICON - Inductive Constraint Programming", contract number FP7-284715.

REFERENCES

[1] J. M. Almendros-Jiménez, 'Querying and reasoning with RDF(S)/OWL in XQuery', in *Proceedings of the* 13^{th} *Web Technologies and Applications Conference (APWeb)*, pp. 450–459, (2011).

[2] A. Bellandi, B. Furletti, V. Grossi, and A. Romei, 'Ontology-driven association rules extraction: a case study', in *Proceedings of the International Workshop on Contexts and Ontologies: Representation and Reasoning (C&O:RR)*, (2007).

[3] T. Guns, S. Nijssen, and L. De Raedt, 'Itemset mining: A constraint programming perspective', *Artif. Intell.*, **175**(12-13), 1951–1983, (2011).

[4] T. Imielinski and H. Mannila, 'A database perspective on knowledge discovery', *Comm. Of The Acm*, **39**(11), 58–64, (1996).

[5] A. Romei and F. Turini, 'XML data mining', *Softw., Pract. Exper.*, **40**(2), 101–130, (2010).

[6] A. Romei and F. Turini, 'Programming the KDD process using XQuery', in *Proceedings of the* 4^{th} *International Conference on Knowledge Discovery and Information Retrieval (KDIR)*, pp. 131–139, (2011).

[7] The Data Mining Group. The Predictive Model Markup Language (PMML). Version 4.1. `www.dmg.org`, 2012.

[8] W3C. OWL Web Ontology Language. W3C Recommendation 10 February 2004. `www.w3.org/TR/owl-features`, 2004.

[9] W3C. XQuery 3.0: An XML Query Language. W3C Working Draft 14 December 2010. `www.w3.org/TR/xquery-30/`, 2010.

Author Index