Efficient Construction of Relational Features

Filip Železný
Czech Technology University in Prague
Technická 2, 166 27 Prague 6, Czech Republic
zelezny@fel.cvut.cz

Abstract

Devising algorithms for learning from multi-relational data is currently considered an important challenge. The wealth of traditional single-relational machine learning tools, on the other hand, calls for methods of 'propositionalization', ie. conversion of multi-relational data into single-relational representations. A major stream of propositionalization algorithms is based on the construction of truth-valued features (first-order logic atom conjunctions), which capture relational properties of data and play the role of binary attributes in the resulting single-table representation. Such algorithms typically use backtrack depth first search for the syntactic construction of features complying to user's mode/type declarations. As such they incur a complexity factor exponential in the maximum allowed feature size. Here we present a polynomial-runtime alternative based on an efficient reduction between the feature construction problem on the propositional satisfiability (SAT) problem, such that the latter involves only Horn clauses and is therefore efficiently solvable.

1 Introduction

Despite the current blossom of relational machine learning (RML) research, the variety of perpetually augmented single-relational, or attribute-value learning (AVL) paradigms still tempts data miners to convert multi-relational data into single-relational representations digestible by AVL algorithms. This kind of representation change is referred to as *propositionalization*, a term originating in the logic-based learning framework adopted in the field of inductive logic programming [3]. Due to the evident infeasibility of a straightforward table-joining approach to propositionalization, sophisticated alternatives have been proposed [4], including aggregation-based strategies, stochastic matching, and others.

This paper follows a method of propositionalization, known as the *extended transformation approach* [5], which

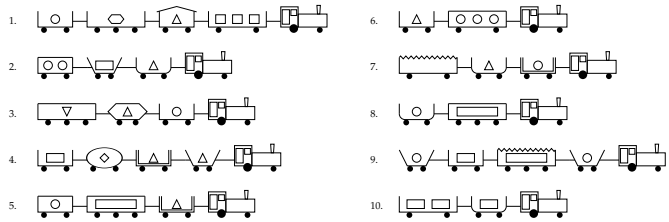

Figure 1. The Michalski train data.

assumes there is a distinct, 'main' table in the original multi-relational database, whose tuples are called individuals. A set of truth-valued relational features is constructed. A feature's value for an individual depends on records spread in the rest of the tables, and related to that individual through foreign key chains. The resulting representation is then a single table, where each feature corresponds to a column, each individual to a row, and each field to the truth-value of the corresponding feature w.r.t the individual. For instance, in the well-known Michalski's east-west trains data set (see Fig. 4) containing a table of trains, a table of cars, and a table of loads, an exemplary relational feature may express that a train (individual) has a car carrying a small, triangle-shaped load. It is convenient to represent features in a simple Datalog-like language, as conjunctions of constant-free, function-free atoms. The mentioned feature then reads

`car(C)` $\wedge$ `load(C,L)` $\wedge$ `small(L)` $\wedge$ `triangle(L)`

When exactly do we consider a conjunction such as the above a well-formed feature? Note, so far informally, a property of the above expression: each variable has exactly one 'output occurrence' and at least one 'input occurrence'. Such expressions we will call *proper*, and we adopt properness as a feature admissibility constraint. For that we need

to assume that arguments in predicates are assigned the respective input/output roles – *modes* prior to feature construction. Furthermore we stipulate that a *type* is declared for each argument place of all predicates, such that no variable may occur at two, differently typed places in an admissible feature. We also require that all admissible features are built out of a pre-defined set of predicates and contain no more than n ($n \in N$) atoms. Finally we dictate that a feature is *connected*: for a proper expression, connectivity means the undecomposability into a conjunction of two admissible features.

Assume now that the user has declared the maximum size n of a feature, the set of available predicates, and has assigned types and input/output modes to arguments places. How difficult is it to *enumerate all (up to variable naming) admissible features*? State-of-art propositionalization algorithms such as RSD or SINUS (see [4] for a description of both) tackle this goal using depth-first backtrack search, starting with an empty expression, gradually adding atoms while maintaining type and mode compliance as well as connectivity, and outputting any found expression of size smaller than n that is proper. The problem of this straightforward strategy is of course its time complexity exponential in n (the depth of the search tree being the maximum feature size). The significance of this problem is emphasized by the fact that often the outlined procedure of syntactic feature construction represents the dominant complexity factor in propositionalization, even diminishing the computational resources needed to find the constructed features' extension (instances for which a given feature holds true).

The main contribution of this paper is in showing an algorithm which, under slight restrictions on the user declaration, (a) finds an admissible feature in polynomial (in n) time and (b) if the declaration allows for at most a polynomial number of admissible features, their entire set will also be constructed in polynomial time (if there are exponentially many admissible features, the algorithm will not introduce an excess exponential factor). Of course, neither (a) nor (b) is the case for the above mentioned backtrack search approaches, which indicates that these state-of-art feature construction algorithms explore some exponentially large subsets of the search space unnecessarily.

Before exposing details, here is a brief outline of our strategy. In all that follows, the adjective *polynomial (exponential)* will stand for *polynomial (exponential) in* n. We first construct a 'bottom feature' $\bot$ – a proper expression (remind the definition of properness from above) complying to the type/mode declaration, which is an atom-wise superset (up to variable renaming) of all admissible features. Slight restrictions on the user declaration will guarantee that $\bot$ exists, has a polynomial number of atoms, and can be constructed in polynomial time. As $\bot$ complies to typing and moding constraints, so do all its subsets. What remains to do then is to find all proper, connected subexpressions of $\bot$ of size $\leq n$, A straightforward verification of all subexpression of size $\leq n$ would obviously require exponential time, however, we show that this problem may be efficiently reduced onto a polynomial-size HORNSAT (satisfiability of propositional Horn clauses) instance, for which an efficient solving algorithm exists. It is interesting to note here that HORNSAT is the only non-trivial, polynomially solvable subclass of the NP-complete SAT problem. Any resulting HORNSAT solution (a truth assignment) can be efficiently translated into an admissible feature. Subsequently we adopt an interesting algorithm producing the entire set of HORNSAT solutions in polynomial time (provided at most a polynomial number of solutions exists) to obtain the entire set of admissible features.

The rest of the paper is organized as follows. The next section details on the above outlined feature construction algorithm. Due to space constraints, we demonstrate here the ideas through examples rather than exposing technical proofs, which will appear in an extended version of this work. Section 3 provides comments on our publicly available Prolog implementation of the presented algorithm. We evaluate it experimentally in Section 4, using two traditional benchmark problems for propositionalization (East-West Trains and Mutagenesis) and compare its performance to a state-of-art propositionalization system. After discussing some open issues and future work in Section 5, we conclude.

2 Method

Remind the basic outline of our feature construction approach from the introduction. We now regard its first step, ie. constructing the bottom feature $\bot$ given a type/mode declaration and n, the maximum feature size. To encode a declaration, we employ a simple form used with slight variations in numerous ILP systems. Here, available predicates are listed with mode and type indicators plugged into the argument places. An example declaration follows

```
car(-c), hasRoof(+c), load(+c,-l),
      triangle(+l), box(+l)
```

Here, the modes `-/+` denote outputs/inputs, respectively, and `c`, `l` represent the respective car and load argument types. We now impose two natural, yet important restrictions on declarations. First, a declaration has a finite size and each declared predicate has a finite arity. Second, there exists a partial irreflexive order $\prec$ on types, such that for any two types t_1, t_2 it holds $t_1 \prec t_2$ whenever t_1 occurs at an input position of a declared predicate and t_2 appears at an output position in the same predicate. This assumption is trivially met by the example declaration above (here `c` $\prec$ `l`). The declaration would remain valid if eg. `draws(+c,+c)`

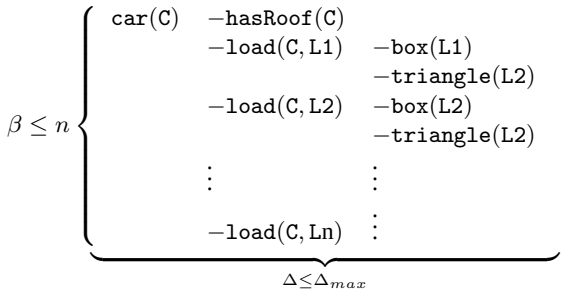

Figure 2. A tree graph representing the bottom feature $\bot$.

was added to it, but not if `draws(+c,-c)` was added. Finally, for clarity of explanation we will only consider predicates with at most one output argument, although the further presented principles do not require that condition.

To demonstrate the construction of $\bot$, we will distinguish two cases: (SI) any declared predicate has at most one input, (MI) some have two or more inputs. We will first exemplify the former case, using the sample declaration above. Due to the $\prec$ existence and the assumption (SI), every admissible feature can be represented as a tree, where vertices correspond to atoms and edges connect pairs of atoms where one contains a variable as an output and the other contains the same variable as an input. Similarly, $\bot$ also corresponds to a tree, which must contain all admissible features as root-sharing subtrees. We sketch the tree form of $\bot$, whose size depends on n, in Fig. 2.

Due to the feature connectivity requirement and assumption (SI), no admissible feature may regard two or more cars: such an expression would necessarily be disconnected.[1] Therefore, only one `car/1` atom is present in $\bot$, as the root. Due to the assumed partial ordering of types $\prec$, the depth of the tree is bounded by some constant Δ_{max}. Also its branching factor can be upper-bounded by n (eg. no feature of size at most n can address more loads than n; this upper bound may of course be quite easily improved). The number of nodes, ie. the size of the bottom set is thus of order $n^{\Delta_{max}}$, ie. polynomial.

Consider now the more general (MI) case where a declaration contains a predicate with multiple inputs. This is a natural case in domains where a feature may relate two substructures of the individual. An example declaration follows capturing a simplified version of the Mutagenesis problem [7].

```
atm(-a), crb(+a), nit(+a), oxy(+a),
hyd(+a),
bond(+a,+a,-b), single(+b), double(+b)
```

With respect to the graph representation we introduced in the previous paragraph, due to the presence of the `bond/3` predicate with 2 inputs, admissible features no longer form a tree and neither does $\bot$.[2] The proof of $\bot$ still having a polynomial size now relies on the fact that $\bot$ can be represented by a directed acyclic graph – its atoms can be organized in 'layers' using the assumed partial type order $\prec$. Up to n atoms are in the first layer, so the first layer generates $O(n)$ output variables. The cardinality of the second layer is thus $O(n^I)$, where I is the maximum number of input arguments in an atom, among atoms in the declaration. The third layer may use $O(n^I)$ variables, so its cardinality is at most $O(n^{I\times I})$. Thus in general, $O(|\bot|)$ may be upper bounded by $n^{I^{\Delta_{max}}}$. Since the exponential factor is a constant, we accept this as a polynomial bound as we intended to achieve in this section. In Section 3, we will avoid this rapid polynomial growth of $|\bot|$ with I and Δ_{max} with small loss in generality by imposing an explicit branching factor bound.

Having constructed $\bot$, we now proceed to the problem of how to efficiently extract all $\bot$'s proper, connected subexpressions of size $\leq n$. The basic idea is to assign a propositional variable to each atom of $\bot$ and use the variables to construct a clause set, encoding the feature admissibility requirements, so that every solution of the clause set corresponds to an admissible feature. Interestingly, encoding the feature constraints turns out to require only Horn clauses. Again, we will illustrate the procedure by way of example in the East-West train domain, continuing with its previous predicate declaration. Let $n = 3$. Then $\bot =$

```
car(C)∧hasRoof(C)∧load(C,L)∧triangle(L)∧box(L)
  P1        P2          P3          P4          P5.
```

is a correct bottom feature. Note that using the branching-factor upper-bound used above for bounding $|\bot|$, we would include three `load/2` atoms into $\bot$ (refer to the corresponding branches in Fig. 2), however, in this case all admissible features of length up to 3 atoms are clearly subsets (up to variable renaming) of this shorter $\bot$. As the lower line indicates, we assign one propositional variable (P_1 to P_5) to each atom. A truth assignment to these variables will represent a $\bot$'s subexpression as follows: if and only if a variable has the *false* value, the corresponding atom *belongs* to the subexpression. As the reader will easily verify, the following set of clauses is satisfied if and only if each variable present in the subexpression has at least one input occurrence (the first clause relating to `C`, the second to `L`).

$$\neg P_2 \vee \neg P_3 \vee P_1 \quad (1)$$

[1] We ignore the case when the declaration has more than one predicate with only output variables (such as `car/1`), while assuming (SI): again due to the connectivity requirement, this case can be treated as two separate feature construction problems.

[2] Also, the `atm/1` predicate will need to be placed n times in $\bot$ with distinct output variables, unlike the `car/1` predicate in the (SI) case.

$$\neg P_4 \vee \neg P_5 \vee P_3 \quad (2)$$

In each clause, we introduced a negative literal corresponding to each atom containing the respective variable as an input, and the positive literal in each clause corresponds to the atom with an output appearance of the respective variable. Since we assume each variable to have exactly one output occurrence, we necessarily obtain Horn clauses. It of course remains to make sure that the mentioned assumption is indeed satisfied. Note first that by construction of $\bot$ (refer to Fig. 2), each output argument is assigned a distinct variable and therefore each variable in any subexpression of $\bot$ appears as an output *at most* once. We now need to make sure that it appears as an output *at least* once. Evidently, this is the case if and only if the following four clauses are satisfied (the upper two for `C`, the lower two for `L`).

$$\neg P_1 \vee P_2 \qquad \neg P_1 \vee P_3 \quad (3)$$
$$\neg P_3 \vee P_4 \qquad \neg P_3 \vee P_5 \quad (4)$$

Here the negative (positive) literals correspond to input (output) occurrences of the respective variables in $\bot$. Since, as we have already seen, there is at most one output occurrence of each variable, also these clauses are necessarily Horn. Let us now determine the total number of Horn clauses obtained in general by the procedure so far. A simple insight yields that we get one clause per every output argument in $\bot$ (such as the two clauses 1 - 2) and one clause per every input in $\bot$ (such as the four clauses 3 - 4). Due to assigning a single propositional variable to every atom in $\bot$, the number of literals in each clause is at most $|\bot|$. As we have constructed a polynomial size $\bot$, the resulting HORNSAT instance (consisting of all clauses 1 - 4) has a polynomial number of clauses with a polynomial number of literals in each. A trivial solution simply makes true all involved propositional variables (note the omnipresence of a positive literal). To avoid this useless solution–corresponding to the empty feature–we append one more Horn clause

$$\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_5 \quad (5)$$

Obviously, there is another trivial solution, which makes all variables false; this case, although unavoidable by means of adding a special clause, does not present a problem as we will see shortly. At this stage, whenever a solution satisfying all clauses constructed so far has n or fewer atoms, it corresponds to an admissible feature. The reader may object that no clause above guarantees the connectivity requirement we have imposed on features. Due to the (SI) character of the particular example at hand, this property is satisfied automatically: from Fig. 2 it is easy to see that any disconnected subgraph of the tree would represent an expression with an input variable with no output occurrence. Such a non-proper expression would be eliminated by the so-far constructed clauses. In the more general (MI) case, however, they indeed do not avoid proper but disconnected expressions[3] such as

$$\texttt{atm(A)} \wedge \texttt{crb(A)} \wedge \texttt{atm(B)} \wedge \texttt{oxy(B)} \quad (6)$$

in the Mutagenesis domain. This problem can be solved by including a polynomial number of additional Horn clauses into the constructed set. The details of this extension are however out of the scope of this paper and will appear in an extended version.

We are now in the position to extract admissible features from $\bot$ by finding a satisfying assignment to a polynomial-size set of propositional Horn clauses. Horn satisfiability was identified as a tractable problem as early as in the 1970's [6] and later, efficient algorithms have been designed [2] able to find a *maximal* solution, that is, one that assigns the *true* value to the greatest possible number of variables, or determine that no solution exists. In our case, a maximal solution corresponds to the smallest connected proper subexpression of $\bot$ (remind that a $\bot$'s atom belongs to the extracted subexpression if its corresponding propositional variable is false). Consequently, if the–efficiently found–maximal solution makes false n or fewer variables, we have found an admissible feature. Otherwise, we can conclude that the declaration allows for no admissible feature. In our continuing example, a maximal solution to the clauses constructed above makes true P_3, P_4 and P_5 (the reader will check that all seven clauses $1-5$ are indeed satisfied), thus P_1 and P_2 are false. This corresponds to the admissible feature `car(C)`$\wedge$`hasRoof(C)`.

Now we can address the objection raised earlier in this section, regarding the trivial satisfiability of the constructed Horn clause by making all variables false: such a solution (corresponding to drawing all atoms from $\bot$) will usually not be maximal, and as such not reported by the solver. Conversely, if it is maximal and $|\bot| \leq n$, then it is actually the only admissible feature. Otherwise, there is no admissible feature.

So far we have merely shown how to efficiently decide the *feature existence* problem by finding an admissible feature if one exists. In practice though, we will need to enumerate the entire set of admissible features. For this purpose, fortunately, we can accommodate the algorithm proposed in [1] able to produce the set of all HORNSAT instance solutions by iterative executions of the core procedure for finding a single solution. The input clause set is at each call modified in a way guaranteeing that the successive

[3] It is easy to show that proper disconnected expressions are conjunctions of admissible features, here `atm(A)`$\wedge$`crb(A)` and `atm(A)`$\wedge$`oxy(A)`. The reason we want to avoid such expression is that AVL algorithms applied on the propositionalization result are themselves usually able to conjugate features, so such decomposable expressions are redundant.

$FeaturesByHORNSAT(\mathcal{D}, n)$: Given an admissible user predicate declaration $\mathcal{D}$ and a number $n \geq 0$, produces the set of all proper connected features of size $\leq n$, satisfying $\mathcal{D}$.

1. Construct bottom feature $\bot = \bot(\mathcal{D}, n)$.
2. Construct HORNSAT instance $\mathcal{H}$ from $\bot$.
3. Find the set $\mathcal{S}$ of all solutions of $\mathcal{H}$.
4. For all $s \in \mathcal{S}$ with at most n false assignments, convert s into the corresponding feature f and output f.

Figure 3. Feature construction through the HORNSAT reduction strategy.

solutions form the entire (lexicographically ordered) set of solutions to the original HORNSAT instance. A favorable property of the algorithm is that the total number of calls to the core procedure is polynomial in (i) the total number of literals in the original clause set, (ii) the number of existing solutions, that is, the algorithm does not introduce an exponential complexity factor when upgrading a single solution finding onto finding of all solutions. We refer the reader to [1] for further details. By employing this algorithm to find all solutions to the HORNSAT instance corresponding to the feature construction problem instance, we do not conduct significant 'excess computation' (corresponding to exploring exponentially large search subspaces containing no solution in the case of standard backtrack feature construction approaches), and specifically, if the actual number of admissible features is polynomial, they are all enumerated in polynomial time.

3 Implementation

We integrated the above described principles into a feature construction system using the YAP Prolog compiler. The Prolog source code can be downloaded from `http://labe.felk.cvut.cz/~zelezny/horn.yap`. The fundamentals steps of the program operation are captured in Fig. 3.

In the next section, we will use the feature construction component of the RSD system [4] as a base-line algorithm. In this system, the user can specify the branching factor β_p for each individual predicate p which has an output by setting its *recall* parameter; for example, a recall of 3 for `load/2` means that at most 3 loads of a given car can be addressed within a single feature. For the theoretical analysis of our algorithm proposed in Section 2, we did not need such predicate-specific bounds (as it sufficed to simply bound each β_p by n). However, we do adopt this way of branching factor bounding from RSD, to enable equivalent feature constraint specification in experiments.

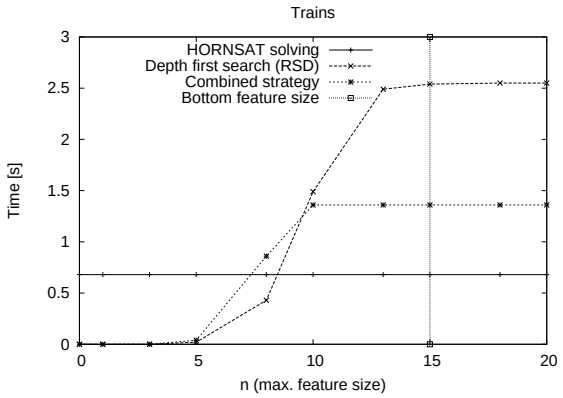

Figure 4. Feature construction run times for Trains.

4 Experiments

Two empirical properties of the proposed approach are of primary interest: (a) the predictive classification accuracy achievable with the features constructed by the algorithm on real-world classification tasks, (b) the runtime performance of the feature-construction algorithm. Given equivalent declarations, the presented algorithm generates the same features as the RSD system. Extensive comparative evaluation of accuracies attainable with such features on real-world classification benchmarks can be found in [4]. The distinguishing point of the present algorithm is the novel, efficient search strategy exploited. We thus do not address (a) here and devote this section to the efficiency evaluation (b).

Although the proposed feature construction algorithm has a polynomial time complexity, it of course incurs certain computational overhead not present in the standard depth first search. In terms of absolute runtime performance, this factor could potentially render the new algorithm inferior to the standard search strategy, albeit of exponential complexity. The purpose of the experiments is thus to verify whether this is the case in two specific tasks.

For the experimentation, we use the two example domains (Trains and Mutagenesis), for which the respective declarations have been shown in Section 2. Because, however, the Train task as declared earlier systematically results in feature construction run times barely distinguishable from 0, we extend the declaration by two predicates `twoWheels(+c)`, `threeWheels(+c)`, yielding measurable times. The branching factors set for the output-containing predicates `car/1`, `load/2`, `atm/1`, `bond/3` were 1, 3, 2, and 1, respectively. The choice of value 1 for `car/1` and `bond/3` was evident: for the former, no connected feature may, under the considered

declaration, address more than one car; for the latter, a bond is of course uniquely identified by the two atoms (predicate inputs) it relates. The other two values were chosen ad hoc, but resulted in effectively measurable running times for a wide range of n values and both systems tested in comparison: RSD, the base-line depth-first search representative, and the newly proposed algorithm based on HORNSAT solving.

Figures 4 and 5 show the measured run times diagrammatically for both systems in the respective tasks. Introducing explicit constant branching factors has three consequences apparent from the figures: (i) the bottom feature size $|\bot|$ is a constant (15 and 22 atoms, respectively) rather than a function of n. We indicate its value on the horizontal axis by means of a vertical line. (ii) The run time of the HORNSAT solving approach is constant in n. That is not surprising if one refers to Fig. 3: for any n, all features subsumed by $\bot$ are first computed and then only those with at most n atoms are kept. (iii) The run time function of depth first search (the RSD system) has an exponential-rate growth approximately within the range of $0 \leq n \leq |\bot|/2$, but then it slows down as $n \rightarrow |\bot|$. This is natural: by definition of $\bot$, there is no feature with n, $n > |\bot|$ atoms, so no search needs to be conducted among expressions of size greater than $|\bot|$. Although RSD does not use $\bot$, its search is constrained by the explicit branching factor bounds with a similar effect. We leave it for the future work to investigate experimentally the more general (although perhaps less natural) setting without explicit branching factor bounds, where the only bound available is n and $|\bot|$ is thus itself a (polynomial) function of n.

Finally, it is no surprise that the depth-first search is much faster for small n (for about $n \leq 7$), while the HORNSAT solving approach is largely superior for greater n (for about $n \geq 10$). We can straightforwardly exploit this fact by means of a robust *combined strategy*, which assigns 50% of cpu time to each strategy and accepts the earlier obtained result. The combined strategy is independent of the particular 'force ratio' change point, and yields the run time twice as high as that of the faster basic strategy for each n (see the corresponding plot in Fig.'s 4 and 5). Of course, more sophisticated ways of combination can be devised, eg. where the proportion of cpu time assigned to the depth-first search decays with n.

5 Conclusion, Future Work

We have presented an algorithm which, under acceptable restrictions on user language declarations, finds a set of admissible declaration-compliant relational features in time polynomial in the maximum feature size, thus improving on the exponential time complexity of state-of-art algorithms.

Our future work plans include the combination of the hereby proposed algorithm for efficient syntactic feature construction with fast algorithms finding the extension of each constructed feature.

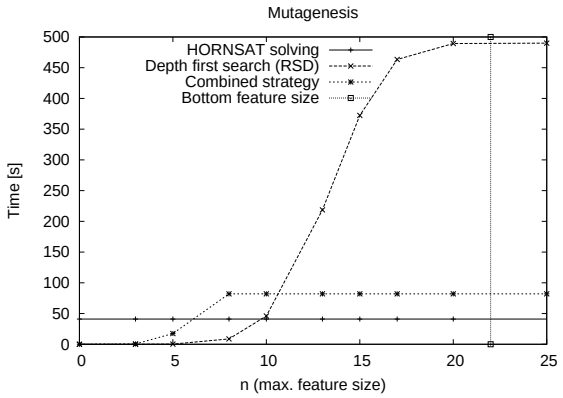

Figure 5. Feature construction run times for Mutagenesis.

Acknowledgments

The research has been supported by the research program No. MSM6840770012 "Transdisciplinary Research in the Area of Biomedical Engineering II" of the CTU in Prague, sponsored by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] R. Dechter and A. Itai. Finding all solutions if you can find one. In *AAAI-92 Workshop on Tractable Reasoning*, 1992.

[2] W. F. Dowling and J. H. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formula. *Journal of Logic Programming*, 3:267–284, 1994.

[3] S. Džeroski and N. Lavrač, editors. *Relational Data Mining*. Springer-Verlag, Berlin, September 2001.

[4] M.-A. Krogel, S. Rawles, F. Zelezny, S. Wrobel, P. Flach, and N. Lavrac. Comparative evaluation of approaches to propositionalization. In *Proceedings of the 13th International Conference on Inductive Logic Programming*. Springer-Verlag, 2003.

[5] N. Lavrač and P. A. Flach. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, 2(4):458–494, October 2001.

[6] T. J. Schaefer. The complexity of satisfiability problems. In *Tenth Annual Symposium on Theory of Computing*, pages 216–226, 1978.

[7] A. Srinivasan, S. H. Muggleton, M. J. E. Sternberg, and R. D. King. Theories for mutagenicity: a study in first-order and feature-based induction. *Artif. Intell.*, 85(1-2):277–299, 1996.