LogCHEM: Interactive Discriminative Mining of Chemical Structure *

Vítor Santos Costa
DCC-FCUP & CRACS
Universidade do Porto, Portugal
vsc@dcc.fc.up.pt

Nuno A. Fonseca
Instituto de Biologia Molecular e Celular (IBMC) & CRACS
Universidade do Porto, Portugal
nf@ibmc.up.pt

Rui Camacho
FEUP & LIAAD
Universidade do Porto, Portugal
rcamacho@fe.up.pt

Abstract

One of the most well known successes of Inductive Logic Programming (ILP) is on Structure-Activity Relationship (SAR) problems. In such problems, ILP has proved several times to be capable of constructing expert comprehensible models that help to explain the activity of chemical compounds based on their structure and properties. However, despite its successes on SAR problems, ILP has severe scalability problems that prevent its application on larger datasets. In this paper we present LogCHEM, an ILP based tool for discriminative interactive mining of chemical fragments. LogCHEM tackles ILP's scalability issues in the context of SAR applications. We show that LogCHEM benefits from the flexibility of ILP, both by its ability to quickly extend the original mining model, and by its ability to interface with external tools. Furthermore, we demonstrate that LogCHEM can be used to mine effectively large chemoinformatics datasets, namely several datasets from EPA's DSSTox database and on a HIV dataset based on the DTP AIDS anti-viral screen.

1 Introduction

One of the most important tasks in chemoinformatics is the task of *structural activity prediction* given a set of small compounds, or drugs, one wants to predict a property of interest. This task can be seen as an instance of a more general task, *structural activity regression* (SAR), where one aims at predicting activity of a compound under certain conditions, given structural data on the compound. Ideally, systems that address this task should be able to identify an *interpretable* discriminative structure which describes the most discriminant structural elements with respect to some target.

In order to build such a system, the first problem that one has to address is how to describe molecules. Coordinate-based representations usually operate by generating features from a molecule's 3D-structure [13]. The number of features of interest can grow very quickly, hence the problem that these systems need to address is how to select the most interesting features and build a classifier from them. Coordinate-free representations can use atom pair descriptors or just the atom-bond structure of the molecule. In the latter case, finding a discriminative component quite often reduces to the problem of finding a Maximum Common Substructure (MCS).

Exact MCS search in a molecule represented as a set of atoms and bonds can be seen as a graph-mining task. In this case, a molecule is represented as a graph $G_M = (V, E)$ where V, the vertices, are atom labels, and E, the edges, are bonds. The search can be improved by adding atom and bond properties. The earliest approaches to search for common substructures or fragments were based on ideas from Inductive Logic Programming (ILP). ILP techniques are very appealing because they are based on a very expressive representation language, first order logic, but they have been criticized for exhibiting significant efficiency problems. As stated by Karwath and De Raedt [11], "their application has been restricted to finding relatively small fragments in relatively small databases". Specialized graph miners have

*This work has been partially supported by by Fundação para a Ciência e Tecnologia and projects STAMPA (PTDC/EIA/67738/2006) and ILP-Web-Service (PTDC/EIA/70841/2006). Nuno A. Fonseca is funded by FCT grant SFRH/BPD/26737/2006.

therefore become quite popular. Systems such as SUBDUE [3] started from the empty graph and then generate refinements either using beam-search or breadth-first. More recent systems such as MoFa [2], gSpan [20], FFSM [9], Gaston [16], and SMIREP [11], use depth-first search, and use compact and efficient representations, such as SMILES, for which matching and canonical forms algorithms exist. Arguably, although such systems differ widely, they all use three main principles: **(i)** only refine fragments that appear in the database; **(ii)** filter duplicates; and **(iii)** perform efficient homomorphism testing.

In this work we present LogCHEM, a tool for discriminative interactive mining of chemical fragments. LogCHEM leverages the flexibility of ILP while addressing the three main principles enunciated above. We demonstrate that LogCHEM can be used to mine effectively large chemoinformatics datasets, such as the DTP AIDS dataset [4]. On the other hand, we demonstrate how LogCHEM can benefit from the flexibility of its representation. LogCHEM can input data from chemical representations, such as MDL's SDF file format, and displays molecules and matching patterns through tools such as VMD [10]. In general, our goal is for LogCHEM to become an ideal system for interactive drug discovery.

The main contributions of this work are as follows. We contribute techniques that allow LogCHEM to mine patterns from large chemical datasets effectively. We show that LogCHEM benefits from the flexibility of ILP, both by its ability to quickly extend the original mining model, and by its ability to interface with external tools.

The rest of the paper is structured as follows. In Section 2 we present and explain the LogCHEM architecture. Section 2.1 introduces the ILP framework used in our experiments. Section 2.2 explains our approach to the problem of mining 2D chemical structures. Section 3 reports on the experiments done and discusses the results obtained. We draw some conclusion in Section 4.

2 The LogCHEM System

LogCHEM system is a tool for discriminative interactive mining of chemical fragments. The interaction with the system is made through a user interface. The system requires two input files: one is a SDF format with atom and bond data on a set of molecules; the other is a file which labels (discriminates) the compounds. We use SDF because it is highly popular and because it can convey 3D structure information. Other formats, such as SML can be translated to SDF through tools such as OpenBabel [6]. Also note that some datasets, such as the DSSTox collection of datasets, include 2D and 3D information in the SDF format.

The input files are processed and given as input to a rule discovery algorithm, that is implemented as an extension of an ILP system (currently Aleph [17]). We significantly improved the ILP search algorithm for this task, as explained in the next section. The ILP engine allows the introduction of extra background knowledge for rule discovery. As an example, we take advantage of this flexibility by allowing the user to introduce well-known molecular structures in the search process. This is supported through a *macro* mechanism where the user provides a pattern which is used to control rule refinement.

The output of the ILP system will be a set of rules, or *theory*. Most often, chemists will be interested in looking at individual rules. LogCHEM first matches the rules against the database, and then allows the user to navigate through the list of matches and visualize visualize them. LogCHEM uses `VMD` [10] to display the molecules and the matching substructures.

2.1 Rule Discovery

The key component of LogCHEM is rule discovery. From a number of ILP algorithms, we chose to base our work on Progol's greedy cover algorithm with Mode Directed Inverse Entailment algorithm (MDIE) [15], as implemented in the Progol, April and Aleph systems [17]. We rely on MDIE to achieve directed search, and we use greedy cover removal as a natural algorithm for finding interesting patterns.

---

induce(B, E)
Input: Background knowledge B, and a finite training set $E = E^+ \cup E^-$.
Output: A set of rules H.
1. $H = \varnothing$
2. **do**
3. $\quad e^+ = Select\ an\ Example\ from\ E^+$
4. $\quad \bot = Saturate(B, e^+)$
5. $\quad h = Search(B, E^+ \cup E^-, \bot)$
6. $\quad H = H \cup h$
7. $\quad E_{covered} = \{e \mid e \in E^+ \wedge B \cup H_i \vDash e\}$
8. $\quad E^+ = E^+ \setminus E_{covered}$
9. **until** $E^+ = \varnothing$
10. **return** H

Figure 1. LogCHEM's default algorithm: MDIE plus Greedy Cover Removal.

The theory induction algorithm we use is shown in Figure 1. The algorithm receives a set of examples $E = E^+ \cup E^-$, where E^+ is the set of positive examples, typically a set of active molecules in LogCHEM, and E^- the set of negative examples, and B the knowledge base. Figure 1 shows the implementation of the standard greedy cover removal algorithm. The algorithm starts from the empty set of rules and tries to explain every example in E^+ by searching for good rules. Each time a new rule h is found (step 5),

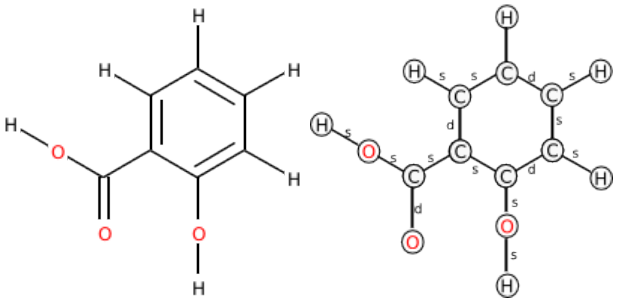

Figure 2. Molecular and graph representation of the chemical compound (Salicylic acid) represented by the SMILE `OC(=O)c1ccccc1O`**.**

the rule h is added to H (step 6), the examples h explains, $E_{covered}$, are removed from the set E (steps $7-8$). The process terminates when every example is explained (step 9) by returning the set of rules H (step 10).

The algorithm presented so far is not specific to ILP. In contrast, steps $3-5$ are specific to ILP and, more specifically, to the MDIE algorithm. Step 3 selects a molecule: by default, Aleph and thus LogCHEM selects the first non-explained molecule. Step 4 is the *saturation* operation: intuitively, it finds a graph, bottom-clause or $\perp$, with the interesting properties of the example. In LogCHEM, we wrote the background knowledge so that $\perp$ is just a logical representation of the undirected graph where the vertices are the atoms and the edges are the bonds between atoms. As an example, consider Figure 2: the saturated clause $\perp$ will include all 16 atoms and 16 bonds. Each bond is represented as two different edges, so the bottom clause will consist of $16+16\times 2=48$ elements. Step 5 is the critical step in this algorithm: it searches for a good clause h by generating and scoring clauses. The process executes until finding the best possible clause, or until reaching an user-predefined limit. As usual, Aleph, and thus LogCHEM, perform general to specific search: they start from very small patterns and refine the patterns by adding extra edges, usually one by one.

2.2 Search in LogCHEM

In order to scale up, a graph mining system must respect three principles: only refine fragments that match the database; avoid duplicates; and, check how many molecules match the patter efficiently. ILP's MDIE provides a natural solution to the first problem: by enumerating patterns from the bottom-clause $\perp$ we guarantee that one example at least will be covered. The second problem is known as redundancy in ILP, and is the main question we address in LogCHEM. The last problem is called coverage calculation in ILP. Most ILP systems rely on Prolog. LogCHEM benefits from recent progress in Prolog technology that allows efficient querying of large datasets of compounds.

2.2.1 Pattern Enumeration

LogCHEM enumerates patterns (or sub-graphs) contained in an example molecule, the seed. LogCHEM builds on an underlying ILP system that enumerates from general to specific. Such systems start from the empty pattern and *refine* it by adding edges or nodes. In the case of an ILP system such as Aleph, and referring back to Figure 2, by default Aleph will start by considering 16 atoms in the molecule, and use each one to generate a pattern; this results in 16 patterns, of which only 3 are different: C, H, and O. In the next step, Aleph system will extend the pattern with more atoms or with more bonds. The process is highly redundant: the pattern $C\vee C$ will be generated in $7\times 6=42$ instances, and the pattern CC will be generated 12 times.

***refine*(π,$\perp$)**
Input: a pattern π and an example's bottom-clause $\perp$
Output: set of patterns Π
1. $\Pi=\emptyset$
2. **forall**$\{Atom \mid Atom\in\pi\}$
3. **while** $Bond=Match(\perp,\pi,Atom)$
4. $\pi'=\pi\cup Bond$
5. **if** $Filter(\pi')$ **then**
6. $\Pi=\Pi\cup\pi'$
7. **return** Π

Figure 3. LogCHEM's refinement algorithm

LogCHEM uses a domain-specific refinement operator designed to generate all contiguous patterns in the molecule. The algorithm is presented in Figure 3. Given an initial pattern π, it returns a set of patterns, Π. Each new pattern π' in Π results from adding an extra edge from the seed molecule to π. This is implemented as follows. First, Π is set to $\emptyset$ (step 1). Next, we consider each atom from the pattern (step 2). Step 3 then searches for a bond $Bond$ which matches the atom $Atom$ and $\perp$ and which is not in π. Step 4 extends the pattern to form a new pattern π'. Step 5 is critical to the performance of LogCHEM: it verifies whether the pattern π' has been generated before. If this is not the case, the algorithm adds π' to Π.

The $Match$ and $Filter$ functions are critical to LogCHEM. It relies on the ILP engine to implement $Match$ through a mechanism known as a user-refinement [17]. The mechanism is extremely efficient because the ILP engine *remembers* how π was generated. More precisely, it maintains how π was embedded in the example molecule. The $Match$ function therefore just has to enumerate open edges in the graph.

2.2.2 Pruning

The $Filter$ function discards redundant rules. While searching for rules, we maintain a store Π_0 with all rules found so far. $Filter$ receives a new rule π' and succeeds if π' is not in the store, and fails otherwise.

Unfortunately, verifying whether two rules match corresponds to the graph homomorphism problem, which is NP-complete. LogCHEM tries to achieve a balance between the cost of finding an exact solution and the cost of allowing redundant patterns. It proceeds in two steps: first, it generates a canonical form; second, it matches the canonical (normal) form against a database Π_0.

LogCHEM uses a variant of Morgan's algorithm to obtain normal forms of molecules [14]. Although Morgan's algorithm is non-deterministic, we try to break ambiguities by exploring information regarding atom's type, the atoms's bonds, and the types of the immediate neighbors. As a last resort, if ambiguities remain, we pick one element arbitrarily.

Pattern Matching Given a new pattern, we are interested in finding out how many molecules support the pattern. ILP systems rely on refutation for this purpose: a pattern matches if the corresponding clause succeeds in the example. However, there is a gap between the patterns LogCHEM generates and the standard usage of unification in logic programming.

To understand the problem, consider the clause:

$$\begin{aligned} active(C) \leftarrow \\ & atom(C, Id_1, \mathtt{c}) \wedge \\ & atom\_bond(C, Id_1, Id_2, \mathtt{c}, \mathtt{n}, 2) \wedge \\ & atom\_bond(C, Id_1, Id_3, \mathtt{c}, \mathtt{n}, 2) \end{aligned}$$

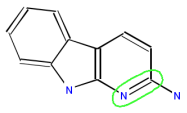

Figure 4. An Example Pattern from a Small Organic Molecule: A-alpha-C.

that represents a $N = C = N$ pattern. Figure 4 matches the molecule A-alpha-C against the pattern. Clearly, there is no match. Unfortunately, Prolog finds a match by matching the same nitrogen against the pattern *twice*. Although this is legitimate in Logic Programming, it is not the intended meaning for patterns.

We would like that different variables match different atoms in the molecule. The standard solution is to rewrite the program to guarantee different variables take different values. Doing so results in the following clause:

$$\begin{aligned} active(C) \leftarrow \\ & atom(C, Id_1, \mathtt{c}) \wedge \\ & atom\_bond(C, Id_1, Id_2, \mathtt{c}, \mathtt{n}, 1) \wedge \\ & Id_1 \neq Id_2 \wedge \\ & atom\_bond(C, Id_1, Id_3, \mathtt{c}, \mathtt{n}, 1) \wedge \\ & Id_1 \neq Id_3 \wedge Id_2 \neq Id_3 \end{aligned}$$

It should be obvious that two variables with different types cannot bind to the same atom. Moreover, two variables on different sides of a bond can never take the same value. If we apply these principles, we get:

$$\begin{aligned} active(C) \leftarrow \\ & atom(C, Id_1, \mathtt{c}) \wedge \\ & atom\_bond(C, Id_1, Id_2, \mathtt{c}, \mathtt{n}, 1) \wedge \\ & atom\_bond(C, Id_1, Id_3, \mathtt{c}, \mathtt{n}, 1) \wedge \\ & Id_2 \neq Id_3 \end{aligned}$$

A second problem concerns pattern evaluation: Prolog will try to build a match step by step. At any point it fails, it will backtrack to the previous match with alternatives. In general, we found out that this works out quite well if the first element in the pattern is unfrequent type, e.g., `Cu` or even `S`, as this reduces the number of hypothesis from the very beginning. Indeed, as LogCHEM by default uses greedy search, experience has shown that most patterns do indeed start from such an element. On the other, if the pattern does not start in this way, execution will be less efficient (we may have to try to match every `C` in a molecule). To ensure that this is taken advantage of, we implemented two pattern optimisation algorithms:

- `BF` chooses the rarest node first and then rewrites the pattern in breadth-first order;
- `DF` also chooses the less frequent atom first, but then generates goals depth-first.

We implement `BF` pattern rewriting by default.

LogCHEM includes a number of further optimisations. Namely, we rewrite bond information in such a way as to minimise backtracking. Also, by default, LogCHEM compiles every pattern, instead of interpreting them, as usual in ILP.

3 Experimental Evaluation

Data and Settings Data for the experiments are from 7 problems. Four datasets were created using data from the EPA's DSSTox database [1]. **CPDBAS** includes information on different chemical properties: we chose to try to predict mutagenicity (other alternatives would be carcinogenic activity in mice and hamsters). **DPBCAN** concerns carcinogenicity of water disinfection by-products. **FDAMDD** dataset concerns predicting the activity category of the maximum daily dosage of chemicals. **NCTRER** concerns FDA National Center for Toxicological Research Estrogen Receptor Binding. Note that we do only use structural-activity relationships and that we do not try to predict QSAR: we only implement the classification task. We did not use EPAFHM due to very high class skew in a small dataset.

The **NCI-HIV** is based on the October 1999 DTP AIDS antiviral screen, that checked tens of thousands of compounds for evidence of anti-HIV activity [4]. Versions of

this dataset have been used in the graph-mining community [12, 11] and for classification purposes [5, 8, 7]. The task we study here is active against inactive; we ignore "somewhat active" compounds.

Finally, the two other datasets are taken from the ILP literature, The **mutagenesis** dataset is a a dataset of mutagenic nitroaromatics [19], and it includes a subset known to be regression friendly. The **carcinogenesis** dataset concerns the well-known problem of predicting carcinogenicity test outcomes on rodents [18].

Table 1 characterizes the datasets in terms of number of positive and negative examples as well as maximum molecules size (number of atoms). The total number of examples ranges from 188 in Mutagenesis regression-friendly (RF) up to 42682 in HIV-A/I. The size of molecules varies widely. As a case in point, almost every molecule in the HIV dataset has at least 20 atoms, 30% have at least 50, and 3% have more than 100 atoms.

Data set	$\mid E^+ \mid$	$\mid E^- \mid$	$\mid MaxAtoms \mid$
mutagenesis RF	125	63	40
mutagenesis	138	92	40
carcinogenesis	182	148	214
CPDBAS	806	738	93
DPBCAN	74	97	26
FDAMDD	575	496	92
NCTRER	96	76	42
HIV-A/I	422	41179	438

Table 1. Data sets (E^+ is the number of positive examples, E^- is the number of negative examples, and $\mid MaxAtoms \mid$ the size of the molecule with most atoms). RF stands for Regression Friendly and A/I for Active/Inactive.

For each application we performed 10-fold cross-validation. We evaluate clauses using *m-estimate* with m being computed automatically: the exceptions were carcino and mutagenesis were we followed common practice and used *compression*. All experiments were run on an AMD Athlon(tm) 64 X2 Dual Core Processor 4600+ with 2GB of SDRAM memory under the Linux operating system and Ubuntu distribution. The machine was being used as a workstation We used YAP-5.1.3 (CVS) [1] compiled under 32-bit mode.

Performance Table 2 shows accuracy and timing results for the benchmarks. The best results were obtained in mutagenesis RF, where accuracy is close to 80%, in DPBCAN, where accuracy is close to 90%, and in NCTRER where accuracy is close to 80%. The worst results were obtained in carcinogenesis, where LogCHEM does not perform much better than default, and in CPDBAS, where accuracy was close to 57%. These results are comparable to results reported by SMIREP on the HIV dataset. Notice that LogCHEM tends to do better in terms of precision than in recall: this is because LogCHEM was set to find precise rules, and we could not find such rules for every example.

The results in Table 2 show that LogCHEM is quite fast for most benchmarks, taking a few seconds to process small benchmarks such as CPDBAS. The system tends to run faster if it can find good rules, as it will need to experiment with less seeds. For example, carcinogenesis and mutagenesis have similar number of examples, but have very different run-times.

Arguably, the AIDS domain is the most challenging one, both in terms of size of molecules and in terms of number of examples. In general, LogCHEM finds rules that are remarkably consistent across folds. Moreover, our results show classifier performance to be similar between training and test set. This suggests that the rules may be generalizing well. Some of the patterns we found have high selectivity and come across most folds (some of them were previously reported in previous work [11]). The pattern was found in molecule m34931: it connects a sulphur with three oxygens to an aromatic ring and thence to a nitrogen and to a carbon opposed. A second common pattern we found links a $N = N$ pair to a furan.

4 Conclusions and Future Work

We present LogCHEM, an integrated system that receives descriptions of molecules and can find interesting discriminative patterns. Our results show that an ILP system is able to find structurally large multi-relational concepts, even in large sets of molecules. The concepts found can be interpreted graphically, and seem to provide some insight into the diverse domains. Moreover, the accuracy of the generated theories is close to state-of-the-art systems.

LogCHEM benefits from the logical representation in a number of ways. Although our representation is less compact than a specialized representation such as as SMILES, used in MOLFEA [12] and SMIREP [11], it offers a number of important advantages. First, it is possible to store information both on atoms and on their location: this is useful for interfacing with external tools. Second, LogCHEM can take advantage of the large number of search algorithms implemented in ILP. Third, given that we implement the basic operations efficiently, we can now take advantage of the flexibility of our framework to implement structured information. We have already taken advantage of this to support *macro* structures, such as rings used in MoFa [2] in a straightforward fashion. Initial results show that LogCHEM does capture rings while maintaining similar running times.

[1] http://www.dcc.fc.up.pt/~vsc/Yap.

Data set	Accuracy	Recall	Precision	Time
mutagenesis RF	0.78 ± 0.10	0.70 ± 0.16	0.94 ± 0.09	38 ± 16
mutagenesis	0.70 ± 0.08	0.56 ± 0.15	0.91 ± 0.06	39 ± 23
carcinogenesis	0.56 ± 0.09	0.47 ± 0.14	0.64 ± 0.12	129 ± 45
CPDBAS	0.57 ± 0.02	0.3 ± 0.05	0.71 ± 0.04	1.3 ± 0.6
DPBCAN	0.89 ± 0.06	0.84 ± 0.13	0.90 ± 0.10	1145 ± 57
FDAMDD	0.65 ± 0.04	0.41 ± 0.09	0.71 ± 0.06	848 ± 205
NCTRER	0.76 ± 0.12	0.69 ± 0.22	0.86 ± 0.10	8.8 ± 8
AIDS-A/I	0.99 ± 0.00	0.45 ± 0.07	0.34 ± 0.06	12257 ± 1207

Table 2. Benchmark Performance of LogCHEM. Accuracy, Recall and Precision are given as fractions. Times are given in seconds.

References

[1] R. AM. DSSTox update & future plans. *QSAR and Modelling Society Newsletter*, 15:34–36, 2004.

[2] C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 51–58. IEEE Computer Society, 2002.

[3] R. N. Chittimoori, L. B. Holder, and D. J. Cook. Applying the subdue substructure discovery system to the chemical toxicity domain. In A. N. Kumar and I. Russell, editors, *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference, May 1-5, 1999, Orlando, Florida, USA*, pages 90–94. AAAI Press, 1999.

[4] J. M. Collins. The DTP AIDS antiviral screen program, 1999.

[5] M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*, pages 35–42. IEEE Computer Society, 2003.

[6] R. Guha, M. T. Howard, G. R. Hutchison, P. Murray-Rust, H. Rzepa, C. Steinbeck, J. K. Wegner, and E. L. Willighagen. The Blue Obelisk–Interoperability in Chemical Informatics. *Journal of Chemical Information and Modeling*, 46:991–998, 2006.

[7] T. Grtner. Predictive graph mining with kernel methods. In *Advanced Methods for Knowledge Discovery from Complex Data*. 2005.

[8] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158–167, New York, NY, USA, 2004. ACM.

[9] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*, pages 549–552. IEEE Computer Society, 2003.

[10] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[11] A. Karwath and L. D. Raedt. Predictive graph mining. In E. Suzuki and S. Arikawa, editors, *Discovery Science, 7th International Conference, DS 2004, Padova, Italy, October 2-5, 2004, Proceedings*, volume 3245 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004.

[12] S. Kramer, L. D. Raedt, and C. Helma. Molecular feature mining in hiv data. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 136–143, New York, NY, USA, 2001. ACM.

[13] G. M. Maggiora, V. Shanmugasundaram, M. J. Lajiness, T. N. Doman, and M. W. Schultz. *A practical strategy for directed compound acquisition*, pages 315–332. Wiley-VCH, 2004.

[14] H. L. Morgan. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113, 1965.

[15] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[16] S. Nijssen and J. N. Kok. Frequent graph mining and its application to molecular databases. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics: The Hague, Netherlands, 10-13 October 2004*, pages 4571–4577. IEEE, 2004.

[17] A. Srinivasan. The Aleph Manual, 2003. Available from `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph`.

[18] A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 273–287. Springer-Verlag, 1997.

[19] A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232, 1994.

[20] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 721–724, 2002.