Avoiding Over-Fitting in ILP-Based Process Discovery

S.J. van Zelst, B.F. van Dongen, W.M.P. van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands
{s.j.v.zelst,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl

Abstract. The aim of process discovery is to discover a process model based on business process execution data, recorded in an event log. One of several existing process discovery techniques is the ILP-based process discovery algorithm. The algorithm is able to unravel complex process structures and provides formal guarantees w.r.t. the model discovered, e.g., the algorithm guarantees that a discovered model describes all behavior present in the event log. Unfortunately the algorithm is unable to cope with exceptional behavior present in event logs. As a result, the application of ILP-based process discovery techniques in everyday process discovery practice is limited. This paper addresses this problem by proposing a filtering technique tailored towards ILP-based process discovery. The technique helps to produce process models that are less over-fitting w.r.t. the event log, more understandable, and more adequate in capturing the dominant behavior present in the event log. The technique is implemented in the ProM framework.

Keywords: Process mining, process discovery, integer linear programming, filtering

1 Introduction

Process mining [1] aims to assist in the improvement and understandability of business processes. The basic input of process mining is process execution data, stored in an *event log*. We identify three process mining branches. *Process discovery* aims at constructing a process model given an event log. *Conformance checking* aims at assessing the conformance of an event log to a given process model. *Process enhancement* aims at extending, improving or repairing existing process models using the two aforementioned disciplines as a basis. In process mining, a process model's quality is evaluated w.r.t. four essential quality dimensions [2]. *Replay fitness* describes to what extent a model is able to reproduce the behavior present in an event log. *Precision* describes what fraction of the behavior allowed by a model is present in an event log. *Generalization* describes to what extent a model is able to reproduce future, unseen, behavior of a process. *Simplicity* describes the (perceived) complexity of a process model.

The ILP-based process discovery algorithm [3] encodes an event log as a set of linear inequalities that act as a core constraint body of a number of integer linear programs (ILPs) aimed at process model construction. The algorithm ensures perfect replay fitness, i.e., all behavior present in the event-log can be reproduced by the resulting process model. Under the assumption that the event log only holds frequent behavior,

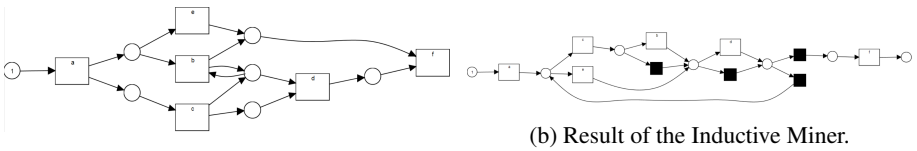
the algorithm works well. Real event logs typically include low-frequent exceptional behavior, e.g. caused by employees deviating from some normative process. As the algorithm guarantees perfect replay-fitness, it guarantees that the resulting model allows for all exceptional behavior present in the event log. In practice this leads to models that are incapable of capturing the dominant behavior present in the event log.

To leverage the strict replay-fitness guaranteed by the ILP-based process discovery algorithm we present a filtering technique that exploits the underlying data abstraction used within the ILP formulation. Using a simple running example we show that the approach enables us to filter exceptional behavior from event logs and results in models that do not have perfect replay-fitness w.r.t. the input data. However, the models are simpler and less over-fitting. To evaluate the technique we have applied it on a set of artificially generated event logs with varying levels of exceptional behavior.

The outline of this paper is as follows. In Section 2 we motivate the need for an ILP-based process discovery algorithm able to cope with the presence of exceptional behavior. In Section 3 we explain the effect of exceptional behavior. In Section 4 we introduce the concept of sequence encodings. In Section 5 we present a sequence encoding based filtering technique. In Section 6 we evaluate the approach in terms of its effects on model quality. Section 7 concludes the paper.

2 Motivation

The ILP-based process discovery algorithm uses Petri nets without arc-weights[1] as a process model formalism. Petri nets allow for expressing complex control flow patterns within event data, a valuable property from a business management perspective. Consider the two models depicted in Figure 1a and Figure 1b which depict the result of applying the ILP-based process discovery algorithm and the Inductive Miner [4] on event log $L = [\langle a, c, d, e, f\rangle^{10}, \langle a, c, b, d, f\rangle^{10}, \langle a, c, e, d, f\rangle^{10}, \langle a, e, c, d, f\rangle^{10}]$[2]. L contains behavior generated by a model exhibiting a *milestone pattern* [5]. The ILP-based discovery algorithm allows us to discover the milestone pattern whereas the inductive miner neglects the pattern and results in an under-fitting Petri net, i.e., it allows for much more behavior compared to the behavior present in the event log. This is due to the fact that the inductive miner assumes that the resulting model is *block-structured*.



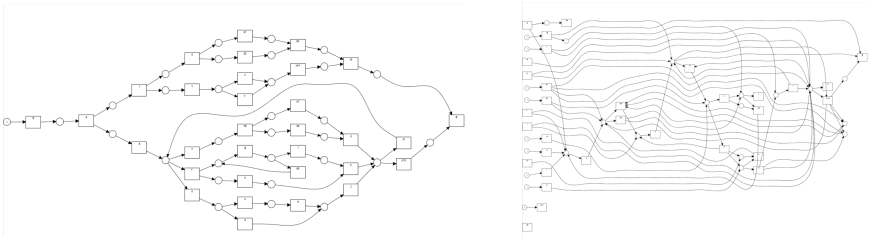
(a) Result of the ILP-based algorithm.

(b) Result of the Inductive Miner.

Fig. 1: Process discovery results of the conventional ILP-based discovery and the Inductive Miner [4] based on a log consisting of milestone pattern based behavior.

[1] We assume the reader to be acquainted with of Petri nets and refer to [1] for an overview.

[2] For event logs we use the notion of a mutliset of traces, using a control-flow perspective.



(a) Resulting model using an event log without exceptional behavior.

(b) Resulting model using an event log with a minimal amount of exceptional behavior.

Fig. 2: Discovered Petri nets after applying conventional ILP-based process discovery on event logs with and without the presence of exceptional behavior in the event log.

Many process discovery algorithms assume models to be (semi)-structured or assume that only local dependencies exist amongst activities. As a side effect, the algorithms are not able to find complex control flow patterns. Examples of such techniques are the Heuristics miner [6], the Fuzzy miner [7] and the Genetic miner [8]. A selection of patterns that the ILP-based process discovery algorithm is able to reproduce are patterns like *interleaved parallel routing*, *critical section* and *arbitrary cycles*.

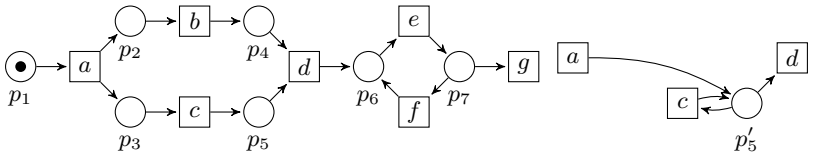
The impact of exceptional behavior present in event logs becomes clear when regarding the two Petri nets depicted in Figure 2. The models are discovered using conventional ILP-based process discovery. The event log used to discover the model in Figure 2a only consists of traces that fit the model presented, i.e. no exceptional behavior. The event log used for discovery of the model in Figure 2b is a slightly manipulated version of the event log used for Figure 2a. The event log contains little exceptional behavior, i.e., 5% of the traces in the event log is manipulated. Clearly, the model depicted in Figure 2b is not capturing the dominant behavior present in the event log.

The ILP-based process discovery algorithm allows for finding complex patterns within business process event data yet at the same time the algorithm suffers drastically from the presence of exceptional behavior in event logs. Therefore we need means to cope with exceptional behavior in order to enable the algorithm to discover models that more accordingly represent the dominant behavior present in an event log.

3 Exceptional Behavior & ILP-Based Discovery

The essential component of the ILP-based process discovery algorithm is a set of linear inequalities, based on the event log, that is used as a basic ILP constraint body. The global constraint expressed by these inequalities is best explained by the following sentence: *Any place present in the resulting Petri net must allow for each event in the input event log to be executed*. This leads to the fact that every trace in the log is completely reproducible by the resulting process model, i.e. replay fitness is perfect. It is also the root cause of the algorithm's behavior w.r.t. to exceptional behavior, e.g. Figure 2.

Consider event log $L = [\langle a, b, c, d, e, g\rangle^{105}, \langle a, c, b, d, e, g\rangle^{98}, \langle a, b, c, d, e, f, e, g\rangle^{87}, \langle a, c, b, d, e, f, e, g\rangle^{117}]$ which could be a result of 407 executions of the process model



(a) Petri net corresponding to example event log L. (b) Place p'_5.

Fig. 3: A Petri net corresponding to event log L, discoverable by the ILP-based process discovery algorithm (3a) and an alternative place p'_5 for place p_5 given L' (3b).

depicted in Figure 3a. Consider place p_5 having an incoming arc from transition c and an outgoing arc to transition d. As place p_5 is not having an outgoing arc to transitions a, b, c, e, f and g it does not interfere with firing these transitions at any point in time. The outgoing arc of place p_5 allows for firing d, only after firing transition c. This is in line with the event log because if event d occurs, it is always (indirectly) preceded by event c. In fact, each place within the Petri net allows for the execution of every activity in L. Hence, the Petri net is discoverable by the ILP-based process discovery algorithm.

Consider the addition of the single trace $\langle a, b, d, e, g\rangle$ to event log L, resulting in L', consisting of 408 traces. Arguably we can deem the newly added trace as exceptional behavior. The newly added trace can not be executed by the Petri net depicted in Figure 3a due to the presence of p_5, as it prevents transition d from firing as long as transition c has not fired. Consequently, given event log L' which only consists of one additional exceptional trace $\langle a, b, d, e, g\rangle$ w.r.t. event log L, the ILP-based process discovery algorithm is unable to find the Petri net of Figure 3a. If we replace p_5 by p'_5 (Figure 3b), the resulting Petri net is again able to execute every activity in L'.

4 Sequence Encoding

The exact construction of the linear inequalities used by the ILP-based process discovery algorithm is outside of the scope of this paper, hence we refer to [3, 9]. It suffices to know that the set of linear inequalities is solely based on the *prefix closure* of the event log. The prefix closure of L' is the set of sequences $\overline{L'}$ s.t. each sequence in $\overline{L'}$ is either a prefix of a sequence in L' or a prefix of a sequence in $\overline{L'}$. Extrapolation of trace frequency information present in the event log to the prefix closure is trivial, i.e., $\overline{L'} = [\epsilon^{408}, \langle a\rangle^{408}, \langle a, b\rangle^{193}, \langle a, c\rangle^{215}, \langle a, b, c\rangle^{192}, \langle a, b, d\rangle, \langle a, c, b\rangle^{215}..., \langle a, b, c, d, e, f, e, g\rangle^{87}, \langle a, c, b, d, e, f, e, g\rangle^{117}]$.

Each non-empty sequence in $\overline{L'}$ is mapped to a linear inequality, representing an ILP constraint. These linear inequalities can be represented by a pair consisting of the Parikh-based multiset representation of the sequence's prefix and the last event of the sequence. Such tuple is deemed a *sequence encoding*. The Parikh-based multiset representation of a sequence is just a mutliset denoting the number of occurrences of each element in the sequence, e.g. given sequence $\langle a, b, b, c\rangle$, its Parikh-based multiset representation is $[a, b^2, c]$. Computing sequence encodings is straightforward, e.g. for $\langle a\rangle$ we have $([], a)$, for $\langle a, b\rangle$ we have $([a], b)$, for $\langle a, c, b, d, e, f, e, g\rangle$ we have $([a, b, c, d, e^2,$

$f], g)$, etc. Different sequences can have the same sequence encoding and therefore map to the same constraint. Consider $\langle a, b, c, d\rangle$ and $\langle a, c, b, d\rangle$ both mapping to $([a, b, c], d)$. The sequence encoding of ϵ is defined as $([], \epsilon)$.

5 Sequence Encoding Filtering

The presence of $\langle a, b, d, e, g\rangle$ in L' causes the ILP-based process discovery algorithm to be unable to find place p_5 of the Petri net depicted in Figure 3a. As the body of constraints of the ILP-based process discovery algorithm, i.e., the set of linear inequalities, specifies this behavior, we need means to remove the inequalities related to $\langle a, b, d, e, g\rangle$ from the constraint body. We do so by constructing a directed acyclic graph where each sequence encoding, i.e., each constraint, acts as a vertex. The sequence encoding $([], \epsilon)$ always acts as a root vertex. Two vertices are connected by an arc if the arc's source could be a prefix of the arc's target. The arcs are labeled using sequence frequencies present in the prefix closure of the event log. An example of such graph, based on $\overline{L'}$ is depicted in Figure 4.

In $\overline{L'}$ the empty sequence ϵ acts as a prefix of $\langle a\rangle$. Hence $([], \epsilon)$ has an outgoing arc to $([], \langle a\rangle)$ labeled with 408, i.e., in 408 cases ϵ acts as a prefix of $\langle a\rangle$. Sequence $\langle a\rangle$ on its turn acts as a prefix of $\langle a, b\rangle$ and $\langle a, c\rangle$, thus, $([], a)$ is connected to $([a], b)$ and $([a], c)$. The edge weights of the arcs from $\langle a\rangle$ to $\langle a, b\rangle$ and $\langle a, c\rangle$ are related to the number of times $\langle a\rangle$ acts as a prefix of $\langle a, b\rangle$, $\langle a, c\rangle$ respectively. Applying the previous rationale on all nodes yields the graph as presented in Figure 4.

After constructing the graph, we traverse it in a breadth-first manner and cut off branches that represent exceptional behavior. We start at its root and assess what outgoing arcs have a too low arc weight given some decision function. Once we have decided what outgoing arcs should remain we traverse each of these arcs. From the end-point of such arc we again evaluate all outgoing arcs. Only those constraints corresponding to a vertex present in the filtered sequence encoding graph will be added to the ILP constraint body. The decision function that decides whether we cut off a certain branch is a parameter of the approach.

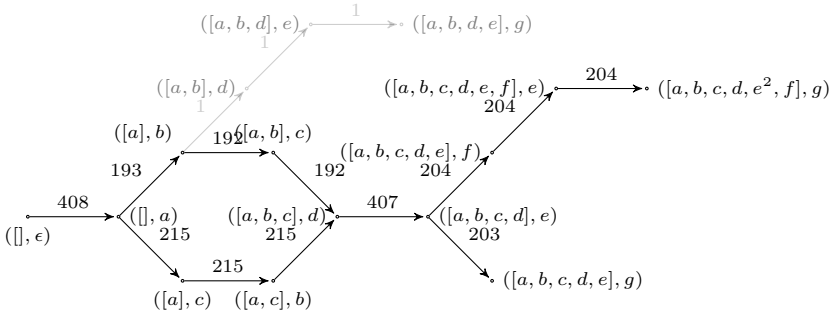


Fig. 4: Sequence encoding graph based on event log $\overline{L'}$. Filtering affected the branch starting at $([a], b)$ and ending in $([a, b, d, e], g)$.

In the implementation of the algorithm we have adopted the following approach. For each vertex that we decide to keep, we always include the outgoing edge with the maximum edge label value. Additionally we include all other edges e that have a lower (or equal) value than (to) the maximum value, as long as the difference of e's value w.r.t. the maximum is within some bounded range. The bounded range is typically some fraction of the maximum, this fraction is deemed the cut-off coefficient c_c. As an example we apply this technique on the graph depicted in Figure 4 with $c_c = 0.25$.

The root has one arc and thus we keep this arc. Traversing the arc leads us to vertex $([], a)$ which has two outgoing arcs. The outgoing arc from $([], a)$ with the maximum label is the arc to $([a], c)$, labeled 215. This arc will be kept in the graph. The bounded range for any other arc starting from vertex $([], a)$ is computed by multiplying the cut-off coefficient with the maximum value for this node, i.e., the bounded range is $0.25 \times 215 = 53.75$. Any edge going out of $([], a)$ that has a value greater than or equal to $215 - 53.75 = 161.25$ is kept in the graph. In this case the arc from $([], a)$ to $([a], b)$ will remain as it has a value of 193. In vertex $([a], b)$ we identify that we keep the edge to $([a, b], c)$, which has the maximum label. We only keep outgoing arcs from $([a, b], c)$ with a label value greater than or equal to $192 - 0.25 \times 192 = 144$. As a result we will drop the edge to $([a, b], d)$ as it only has a label value of 1. The result of repeating the filtering procedure on all vertices is visualized Figure 4 where the filtered branch, i.e., starting at $([a], b)$ and ending in $([a, b, d, e], g)$, is graying out. Note that using the aforementioned approach all constraints related to (prefixes of) $\langle a, b, d, e, g\rangle$ are remove from the constraint body. As a consequence, place p_5 in Figure 3a becomes a feasible place again. Thus, the model depicted in Figure 3a can be found using sequence encoding filtering applied on L'. As the model does not allow for $\langle a, b, d, e, g\rangle$, we no longer guarantee perfect replay fitness w.r.t. L'.

6 Evaluation

For evaluation we used an implementation of sequence encoding filtering present in the *HybridILPMiner*[3] package within the *ProM framework* (`http://www.promtools.org`). Here we discuss effects on model quality. For a quantification of the effect on ILP solve time we refer to [9]. The event logs used for evaluation are artificially generated event logs originating from a study related to the impact of exceptional behavior on rule-based approaches in process discovery [10]. The event logs contain different levels of exceptional behavior and are based on two ground truth event logs. The ground truth event logs, *a22f0n00* and *a32f0n00*, are free of exceptional behavior, i.e. all traces fit the originating model. *a22f0n00* consists of 22 different event classes whereas *a32f0n00* consists of 32 different event classes. Out of each ground truth event log a total of four new logs is generated, consisting of either 5%, 10%, 20% or 50% of manipulated traces. Manipulation of traces is performed by either tail/head of sequence removal, random part of sequence body removal or interchange of two randomly chosen events [10].

We primarily focus on precision, i.e. the amount of behavior allowed by the model also present in the event log. If all behavior allowed by the model is present in the event

[3] `https://svn.win.tue.nl/repos/prom/Packages/HybridILPMiner/Branches/experiments/2015_bpm_ilp_filtering-0.2.1/`

log, precision is maximal and equals 1. The more behavior is allowed by the model that is not present in the event log, the lower the precision value will be. By definition, the conventional ILP-based process discovery algorithm will result in models that allow for all behavior present in the event log. Thus, if we use the conventional algorithm on a manipulated event log, the resulting model will allow for all exceptional behavior. As the exceptional behavior is not present in the ground truth events log, computing precision of the resulting model based on the ground truth log is expected to be low. On the other hand, if we discover models using an algorithm that is more able to handle the presence of exceptional behavior, we expect the algorithm to allow for less exceptional behavior and, w.r.t. the ground truth model, we expect a higher precision value.

In Figure 5 the results of applying the conventional ILP-based process discovery algorithm and three different sequence encoding filtering instantiations for each event log are depicted. We used the branch cut-off technique as described in Section 5 with cut-off coefficients $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$. We measured both the replay-fitness and precision based on the ground truth event logs. Replay-fitness of the discovered models w.r.t. the ground truth event logs using all four approaches remains 1 in all cases[4]. Due to the incapability of handling exceptional behavior of the conventional algorithm, as expected, precision drops rapidly. For the sequence encoding filtering we identify the $\frac{1}{4}$ variant to outperform the other two. This is explained by the fact that this variant is the most rigorous filter and removes the most constraints. It is clear that the decrease of precision for the sequence encoding based approaches is less severe compared to the conventional approach. This is in line with the rationale presented before, i.e., we expect the filtering based approaches to be more able in handling exceptional behavior. Therefore, we conclude that the filtering based models discover Petri net patterns that more accurately represent the dominant behavior in the input event log. Thus, the newly presented tech-

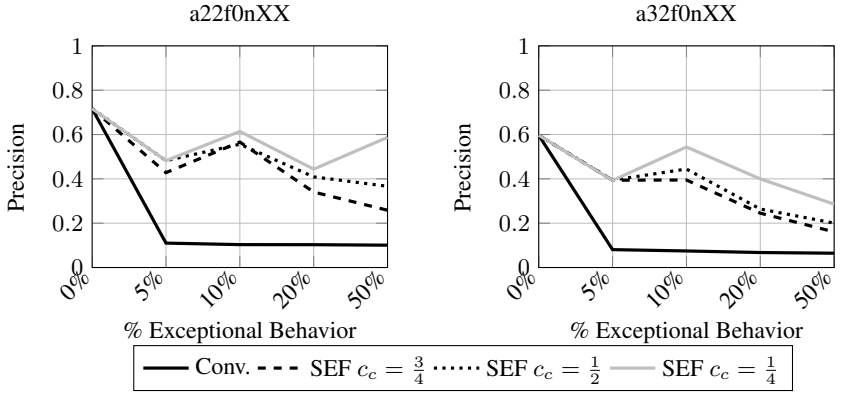


Fig. 5: Precision measurements based on event logs with exceptional behavior. Conventional ILP-based process discovery (Conv.) and sequence encoding filtering (SEF) was used for porcess model discovery.

[4]One exception for SEF with $c_c = \frac{1}{2}$, where replay fitness equals 0.99515.

niques allow us to successfully apply filtering whilst using ILP-based process discovery as a basis.

7 Conclusion

The work presented in this paper is motivated by the observation that the existing ILP-based process discovery algorithm is unable to cope with exceptional behavior in event logs. ILP-based process discovery has several advantages, but the inability to abstract from infrequent exceptional behavior makes it unusable in real-life settings. We presented the sequence encoding filtering technique which enables us to apply filtering exceptional behavior within the ILP-based process discovery algorithm. The technique allows us to find models with acceptable trade-offs w.r.t. replay fitness and precision. We showed that the technique enables us to find Petri net structures in data consisting of exceptional behavior, using ILP-based process discovery as an underlying technique.

References

1. Aalst, W.M.P.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F., eds.: On the Move to Meaningful Internet Systems: OTM 2012. Volume 7565 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 305–322
3. Werf, J.M.E.M.v.d., Dongen, B.F.v., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundamenta Informaticae **94**(3) (2009) 387–412
4. Leemans, S.J.J., Fahland, D., Aalst, W.M.P.v.d.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings. (2013) 311–329
5. Aalst, W.M.P.v.d., Hofstede, A.H.M.t., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
6. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France. (2011) 310–317
7. Günther, C.W., Aalst, W.M.P.v.d.: Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In: Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings. (2007) 328–343
8. Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: A Genetic Algorithm for Discovering Process Trees. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012. (2012) 1–8
9. Zelst, S.J.v., Dongen, B.F.v., Aalst, W.M.P.v.d.: Filter Techniques for Region-Based Process Discovery. Technical Report 15-4, BPM Center.org (2015)
10. Maruster, L., Weijters, A.J.M.M., Aalst, W.M.P.v.d., Bosch, A.v.d.: A Rule-Based Approach for Process Discovery: Dealing with Noise and Imbalance in Process Logs. Data Min. Knowl. Discov. **13**(1) (2006) 67–87