Automatic Determination of Parameters' Values for Heuristics Miner++

Andrea Burattin, *Student Member, IEEE* and Alessandro Sperduti, *Senior Member, IEEE*

***Abstract*— The choice of parameters' values for noise-tolerant Process Mining algorithms is not trivial, especially for users that are not expert in Process Mining. Exhaustive exploration of all possible set of values is not feasible, since several parameters are real-valued. Selecting the "right" values, however, is important, since otherwise the control-flow network returned by the mining can be quite far from the correct one. Here we face this problem for a specific Process Mining algorithm, i.e. Heuristics Miner++. We recognize that the domain of real-valued parameters can be actually partitioned into a finite number of equivalence classes and we suggest exploring the parameters space by a local search strategy driven by a Minimum Description Length principle. We believe that the proposed approach is sufficiently general to be used for other Process Mining algorithms. Experimental results on a set of randomly generated process models show promising results.**

I. Process Mining

Process mining is the area of data mining that extracts important information from data that refers to business processes. A business process is a sequence of steps performed for a given purpose. Examples of these processes are "the software development process" or the "management of an order from a supplier".

Nowadays, the diffusion of IT systems for supporting the execution of business processes, allows many companies to gather a large amount of data recording the history of the performed activities. These logs contain important information, since in principle they could be used to reconstruct the actual control-flow of work. A manual analysis of the logs, however, is not feasible due to the large size of such logs. The aim of Process Mining algorithms is to try to automate this analysis.

Within process mining there are three types of algorithms that can be distinguished (as presented also in [1]): *i)* those for *control-flow discovery*, that try to build a model (such as a Petri Network) describing the behaviour of the process; *ii)* those for *conformance analysis*, that starting from a process model and a log file, determine how much the behaviour described in the log reflects that of the model; *iii)* and finally, there are the *organizational mining algorithms* that, assuming the log contains information about the author of each action, extract a "social network" that describes the set of relations between actions' authors.

In this paper, we present results referring to control-flow discovery algorithms. As an example, consider the log W:

$$W = \left\{(A, B_1, B_2, C, D)^5 \ ; \ (A, B_2, B_1, C, D)^5\right\}$$

The authors are with the Department of Pure and Applied Mathematics, University of Padua, Italy (email: burattin@math.unipd.it, sperduti@math.unipd.it).

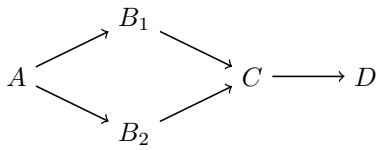

Fig. 1. An example of mined process model for the log W, where activities B_1 and B_2 are executed in parallel.

where we have 10 process instances (5 repetitions of the first type and 5 for the second) that differ in the ordering of the activities B_1 and B_2. From this log, an hypothetical process mining system can infer the presence of 5 activities and, since the ordering between B_1 and B_2 is not always the same through the log, we can assume their execution to be "parallel". In Fig. 1 we show the result of a typical process mining algorithm: the start activity must be A, followed by the parallel execution of activities B_1 and B_2. When B_1 and B_2 terminate, activity C can be executed, followed by activity D.

When considering real-world industrial scenarios, however, it is hard to have the availability of a complete log for a process. In fact, the most typical case is the one where the log is partial and/or noisy.

A log is *partial* if it does not contain a record for all the performed activities; it is *noisy* if either:

1) some recorded activities do not match with the "expected" ones, i.e. there exist records of performed activities which are unknown or which are not expected to be performed within the business process under analysis (for example an activity that, in a real environment is required, but that is unknown to the designer);
2) some recorded activities do not match with those actually performed, i.e. activity A is performed, but instead of generating a record for activity A, a record for activity B is generated; this error may be introduced by a bug into the logging system or due to the unreliability of the transmission channel (e.g. a log written to a remote place).
3) the order in which activities are recorded may not always coincide with the order in which the activities are actually performed; this may be due to the introduction of a delay in recording the beginning/conclusion of the activity, e.g. if the activity is a manual activity and the worker delays the time to record the start/end of the activity, or to delays introduced by the transmission channel used to communicate the start/end of a remote activity.

While case 1) may be acceptable in the context of workflow discovery, where the names of the performed activities are not set or known a priori, cases 2) and 3) may clearly interfere with the mining of the process, leading to an incorrect control-flow reconstruction (that is a control-flow different from the one that the process designer would expect). Because of that, it is important for a process mining algorithm to be noise-tolerant. This is especially true for the task of control-flow discovery, where it is more difficult to detect errors because of the initial lack of knowledge on the analysed process.

A well known example of noise-tolerant, control-flow discovery algorithm is Heuristics Miner, described in [2], [3]. Heuristics Miner has been successively generalized to the treatment of time intervals by Burattin & Sperduti in [4] (Heuristics Miner++), in order to exploit the information about the duration of each activity, when this information is actually available. A typical problem that users face, while using Heuristics Miner, is the need to set the values of specific real-valued parameters[1] which control the behaviour of the mining, according to the amount and type of noise the user believes is present into the process log. Since the algorithm constructs the model with respect to the number of observations in the log, its parameters are acceptance thresholds on frequencies of control-flow relevant events observed into the log: if the observed event is frequent enough (i.e. its frequency is above the given threshold for that event) then a specific feature of the control-flow explaining the event is introduced. Different settings for the parameters usually lead to different results for the mining, i.e. to different control-flow networks.

While the introduction of these parameters and its tuning is fundamental to allow the mining of noisy logs, the unexperienced user may find difficult to understand the meaning of each parameter and the effect, on the resulting model, of changing the value of one or more parameters from one value to another one. Sometimes, even experienced users find it difficult to decide how to set these parameters.

In this paper, we propose to face the problem of parameters tuning for the Heuristics Miner++ algorithm. The approach we adopt starts by recognizing that the domain of real-valued parameters can be actually partitioned into a finite number of equivalence classes and we suggest to explore the parameters space by a local search strategy driven by a Minimum Description Length principle. The proposed result is then tested on a set of randomly generated process models, obtaining promising results.

II. Heuristics Miner++

In the following section we are going to recall the main features, introduced by Heuristics Miner++, that are used by the mining and that allow the construction of the process control-flow model.

[1]Since Heuristics Miner++ is a generalization of Heuristics Miner and since it is "backward compatible", it uses the same set of parameters as Heuristics Miner.

1) Dependency Relations ($\Rightarrow$): An edge (that usually represents a dependency relation) between two activities is added if its *dependency measure* is above the value of the *dependency threshold*. This relation is calculated, between activities X and Y, as:

$$X \Rightarrow_W Y = \frac{|X >_W Y| - |Y >_W X|}{|X >_W Y| + |Y >_W X| + 2|X \|_W Y| + 1} \quad (1)$$

where $|X >_W Y|$ indicates the number of times that X precedes Y in the log W and $|X\|_W Y|$ indicates the number of times that X is overlapped with Y. The rationale of this rule is that two activities are in a dependency relation if most of times they are in the specifically required order and if they are not overlapped (if two activities' time spans are overlapped this mean they are executed in parallel).

2) AND/XOR Relations ($\wedge$, $\otimes$): When an activity has more than one outgoing edge, the algorithm has to decide whether the outgoing edges are in AND or XOR relation ("type of split"). Specifically, it calculates the following quantity:

$$X \Rightarrow_W (Y \wedge Z) = \frac{|Y >_W Z| + |Z >_W Y| + 2|X \|_W Y|}{|X >_W Y| + |X >_W Z| + 1} \quad (2)$$

If this quantity is above a given *AND threshold*, the split is an AND-split, otherwise it is considered to be in XOR relation. The rationale of this case is that two activities are in an AND relation if most of times they are observed in no specific order (so one before the other and vice versa), of if they are explicitly observed as overlapped.

3) Long Distance Relations ($\Rightarrow^l$): Two activities X and Y are in long distance relation if there is a dependency between them, but they are not in direct dependency. This relation is expressed by the formula:

$$X \Rightarrow^l_W Y = \frac{|X \ggg_W Y|}{|Y| + 1} \quad (3)$$

where $|X \ggg_W Y|$ indicates the number of times that X is directly or indirectly (there are other different activities between X and Y) followed by Y in the log W. If this formula's value is above a *long distance threshold*, then a long distance relation is added into the model.

4) Loops of Length one and two: A loop of length one (i.e. a self loop on the same activity) is introduced if the quantity

$$X \Rightarrow_W X = \frac{|X >_W X|}{|X >_W X| + 1} \quad (4)$$

is above a *length-one loop threshold*. A loop of length two is considered differently: it is introduced if the quantity:

$$X \Rightarrow^2_W Y = \frac{|X >^2_W Y| + |Y >^2_W X|}{|X >^2_W Y| + |Y >^2_W X| + 1} \quad (5)$$

is above a *length-two loop threshold*. In this case, the $X >^2_W Y$ relation is observed when X is directly followed by Y and then there is X again.

A. Parameters of the algorithm

In addition to the thresholds introduced above, Heuristics Miner++ uses few more threshold parameters. Let us give below a short description for all of them, so to have a complete list to refer to.

a) Relative-to-best threshold: this parameter indicates that we are going to accept the current edge (i.e. to insert the edge into the resulting control-flow network) if the difference between the value of the dependency measure computed for it and the greatest value of the dependency measure computed over all the edges is lower than the value of this parameter.

b) Positive observations threshold: with this parameter we can control the minimum number of times that a dependency relation must be observed, between two activities: the relation is considered only when this number is above the parameter's value.

c) Dependency threshold: this parameter is useful to discard all the relations whose dependency measure is below the value of the parameter.

d) Length-one loop threshold: this parameter indicates that we are going to insert a length-one loop (i.e. a self loop) only if the corresponding measure is above the value of this parameter.

e) Length-two loop threshold: this parameter indicates that we are going to insert a length-two loop only if the corresponding measure is above the value of this parameter.

f) Long distance threshold: with this parameter we can control the minimum value of the long distance measure in order to insert the dependency into the final model.

g) AND threshold: this parameter is used to distinguish between AND and XOR splits (if there is more than one connection exiting from an activity): if the AND measure is above (or equal) to the threshold, an AND-split is introduced, otherwise a XOR-split is introduced.

In order to successfully understand the next steps, let's point out an important observation: by definition, all these parameters can have values between -1 and 1 or between 0 and 1. Only the positive observation threshold requires an integer value that expresses the absolute minimum number of observations.

Heuristics Miner++, as first step, extracts all the required information from the process log; then it uses the threshold parameters described above. Specifically, before starting the control-flow model mining, it creates the following data structures, where $\mathcal{A}_W$ is the set of activities contained in the log W:

- $directSuccessionCount$, a matrix of size $|\mathcal{A}_W|^2$;
- $parallelCount$, a matrix of size $|\mathcal{A}_W|^2$;
- $dependencyMeasures$, a matrix of size $|\mathcal{A}_W|^2$;
- $L1LdependencyMeasures$, a vector of size $|\mathcal{A}_W|$;
- $L2LdependencyMeasures$, a matrix of size $|\mathcal{A}_W|^2$;
- $longRangeDependencyMeasures$, a matrix of size $|\mathcal{A}_W|^2$;
- $andMeasures$, a matrix of size $|\mathcal{A}_W|^2$.

All the entries of the above data structures are initialized to 0. Then, for each process instance (usually called *case*, in process mining) registered into the log, if two activities (a_i, a_{i+1}) are in a direct succession relation, the value of $directSuccessionCount[a_i, a_{i+1}]$ is incremented, while if they are executed in parallel (i.e., the time intervals associated to the two activities overlap) the value of $parallelCount[a_i, a_{i+1}]$ is incremented; moreover, for each activity a, Heuristics Miner++ calculates the length-one loop measure $a \Rightarrow_W a$, by Eq. (4), and adds its value to $L1LdependencyMeasures[a]$. Then, for each activities pair (a_i, a_j) Heuristics Miner++ calculates the following:

- the dependency measure $a_i \Rightarrow_W a_j$, by Eq. (1), and adds its value to $dependencyMeasures[a_i, a_j]$. It must be noticed that in order to compute Eq. (1) the values $|a_i \overline{>}_W a_j|$ and $|a_i \|_W a_j|$ must be available: these values corresponds to the values found in $directSuccessionCount[a_i, a_j]$ and $parallelCount[a_i, a_j]$, respectively;
- the long distance relation measure $a_1 \Rightarrow^l_W a_2$, by Eq. (3), and adds its value to $longRangeDependencyMeasures[a_1, a_2]$;
- the length 2 loop measure $a_1 \Rightarrow^2_W a_2$, by Eq. (5), and adds its value to $L2LdependencyMeasures[a_1, a_2]$.

Finally, for each triple (a_1, a_2, a_3) the procedure calculates the AND/XOR measure $a_1 \Rightarrow_W (a_2 \wedge a_3)$, by Eq. (2), and adds its value to $andMeasures[a_1, a_2, a_3]$.

When all these values are calculated, Heuristics Miner++ proceeds to the real control-flow construction. These are the main steps: first of all, a node for each activity is inserted; then, an edge (i.e. a dependency relation) between two activities a_i and a_j is inserted if the entry $dependencyMeasures[a_i, a_j]$ satisfies all the constraints imposed by thresholds *a)*, *b)*, and *c)*.

The algorithm continues iterating through all the activities that have more than one connection exiting from it: it is necessary to disambiguate the split behaviour between a XOR and an AND. In these cases (e.g., activity a_i has two exiting connections with activities a_j and a_k), Heuristics Miner++ checks the entry $andMeasures[a_i, a_j, a_k]$ and, if it's above the *AND threshold*, it is marked as an AND-split, otherwise as a XOR-split. If there are more than two activities in the "output-set" of a_i then all the pairs are checked.

A similar procedure is used to identify length-one loop: Heuristics Miner++ iterates through each activity and checks in the $L1LdependencyMeasures$ vector if the corresponding measure is greater than the *d)* threshold. For the length-two loops the procedure checks, for each activities pairs (a_i, a_j), if $L2LdependencyMeasures[a_i, a_j]$ satisfies the *e)* threshold and, if necessary, adds the loop.

The same process is repeated even for the long distance dependency: for each activity pairs (a_i, a_j), if the value of $longRangeDependencyMeasures[a_i, a_j]$ is above the value of the *Long distance threshold* parameter, then the dependency between the two activities is added.

Once Heuristics Miner++ has completed all these steps, it can return the final process model. In this case, the final

model is expressed as a Heuristics Net (an oriented graph, with information on edges, that can be easily converted into a Petri Net).

III. Facing the Parameters Setting Problem

As already said, it is not easy for a user (typically process miner users are business process managers, resources managers, or business unit directors) to decide which values to use for the parameters described above: he/she is not an expert in process mining, and anyway, also an experts in process mining may have an hard time to figure out which setting makes more sense.

The main issue that makes this decision difficult is the fact that almost all parameters take values in real-valued ranges: there is an infinite number of possible choices! Moreover, how can it be possible to select the "right" value for each parameter? Is it preferable to set the parameters so as to generate a control-flow network able to explain all the cases contained in the log (even if the resulting network is very complex and thus hard to understand by a human), or a simpler, and so more readable, model (even if it does not explain all the data)?

Here we assume that the user's desired result of the mining is a "sufficiently" simple control-flow network able to explain as many as possible cases contained in the log. In fact, if the log is noisy, a control-flow network explaining all the cases is necessarily very complex because it has to explain also the noise itself (see [5], for a discussion on this issue).

On the basis of this assumption, we suggest addressing the parameters setting problem by a two step approach:

1) identification of the *candidate hypothesis*, that correspond to the assignments of values to the parameters that induce Heuristics Miner++ to produce different control-flow networks;
2) *exploration* of the hypothesis space to find the "best solution", i.e. generation of the simplest control-flow network able to explain the maximum number of cases.

The aim of step 1) is to identify the set of different process models which can be generated by Heuristics Miner++ by varying the values of the parameters. Among these process models, the aim of step 2) is to select the process model with the best trade-off between complexity of the model description and number of cases that the model is not able to explain. Here, our suggestion is to use the Minimum Description Length (MDL) [6] approach to formally identify this trade-off. In the next two sections, we describe in detail our definition of these two steps.

IV. Discretization of the parameters' values

As discussed in the previous section, by definition, most of Heuristics Miner++ parameters can take an infinite number of values. In practice, only some of them produce a different model as output. In fact, the size of the log used to perform the mining can be assumed to be finite, and thus Equations (1)-(5) can return only a finite number of different values. These sets, with all the possible values, are obtained by calculating the results of the formulas against all single activities (Eq. (4)), all pairs (Eq. (1), (3), (5)) and all triples (Eq. (2)). Specifically, if we look at the data structures used by Heuristics Miner++ and described in Section II, these data structures are populated with all the results just described so they contain all the possible values of the measures of interest for the given log. Even considering the worst case, i.e. when each activity configuration has a different measure value, the mining algorithm cannot observe more than $|\mathcal{A}_W|^i$ different values for parameters described by a i-dimensional matrix. Since $|\mathcal{A}_W|$ is typically a quite low value, even the worst case does not produce a huge number of possible values. Thus it does not make sense to let the thresholds to assume any real-value in the associated range.

Given a log W, let sort, in ascending order, all the different values $v_1, \ldots, v_s$, that a given measure can take. Then, all the values in the ranges $[v_i, v_{i+1})$ with $i = 1, \ldots, s$ constitute equivalence classes with respect to the choice of a value for the threshold associated to that measure. In fact, if we pick any value in $[v_i, v_{i+1})$, the output of the mining, i.e. the generated control-flow network, is not going to change. If the parameters were independent, it would be easy to define the set of equivalence classes. In fact, given n independent parameters $p_1, \ldots, p_n$ with domains $D_1, \ldots, D_n$, it is sufficient to compute the set of equivalence classes $\mathcal{E}_{p_i}$ for each parameter p_i, and then obtain the set of equivalence classes over *configurations of the n parameters* as the Cartesian product $\mathcal{E}_{p_1} \times \mathcal{E}_{p_2} \times \ldots \times \mathcal{E}_{p_n}$. This means that we can uniquely enumerate process models by tuples $(d_{1,i_1}, \ldots, d_{n,i_n})$, where $d_{j,i_j} \in D_j,\ j = 1, \ldots, n$.

Unfortunately, by definition, Heuristics Miner++ parameters are not independent. This is clearly exemplified by considering only the two parameters "positive observation threshold" and "dependency threshold". If the first one is set to a value that does not allow a particular dependency relation to compare in the final model (because it does not occur frequently enough in the log), then, there is no value for the dependency threshold, involving the excluded dependency relation, that will modify the final model. As shown in the example, the lack of independence entails that the mining procedure may generate exactly the same control-flow network starting by different settings for the parameters. This means that it is not possible to uniquely enumerate all the different process models by defining the equivalence classes over the parameters values as discussed above under the independence assumption. So, since there is not a bijective function between process models and tuples of discretized parameters, it is not possible to efficiently search the "best" model by searching among the discretized space of parameters. However, discovering all the dependences among the parameters and then defining a restricted tuple space where there is a one to one correspondence between tuples and process models would be difficult and expensive. Therefore, we decided to adopt the independence assumption to generate the tuple space, while using high level knowledge about the dependences among parameters to factorize the

tuple space in order to perform an approximate search. This feature will be discussed in the next section.

As a final remark, let us give some implementation details about the discretization of the parameters. After the calculation of all formulas values (and the $directSuccessionCount$ matrix in order to cope with discretization of parameter (b)), we implemented a procedure that adds the computed values into an HashSet (one HashSet per parameter).[2] When this calculation is finished, each HashSet will contain all the possible different values, for each parameter, referring to the input log W. This discretization process does not alter the time complexity of the algorithm, however it augments the space required: it has to store all the different measure values for each matrix which, in the worst case, is $|\mathcal{A}|+4|\mathcal{A}|^2+|\mathcal{A}|^3$ (but remember that $|\mathcal{A}|$, usually, is small). Let us recall, now, the list of steps done in this first phase:

1) calculation of the measure values for all the possible activities configuration;
2) from these values we can populate the various matrices used by Heuristics Miner++;
3) from the same set of values we can also compute the HashSet with the "discretized parameter set".

V. Exploration of the Hypothesis Space

What we have just described is a possible way to construct a set with all values, for each parameter, that produce distinct process models. As we have discussed in Section III, each process model mined from a particular parameters configuration constitutes, for us, a hypothesis (i.e. a potential candidate to be the final process model). We are, now, in this situation:

- we can build a set with all possible parameters values;
- each parameters configuration produces a process model hypothesis.

Starting from these two elements, we can realize that we have all the information required for the construction of the hypothesis space: if we enumerate all the tuples of possible parameters configurations (and this is possible, since these sets are finite) we can build the set of all possible hypothesis, that is the hypothesis space. The second step, described in our approach, requires the exploration of this space, in order to find the "best" hypothesis.

As we have already explained, the parameters discretization process does not produce a large number of possible values but, since the hypothesis space is given by the combination of all the parameters' values, this can become quite large, and finding the best hypothesis easily turns into a quite complex search problem: an exhaustive search of the hypothesis space (that will lead to the optimal solution) is not feasible. So we decided to factorize the search space by exploiting high level knowledge about independence relations (total and conditional) among parameters, and to explore the factorized space by a local search strategy. Let start by describing how we factorized the search space.

[2]The whole procedure has been written in Java language and encoded as a plugin of the ProM Framework [7].

A. Factorization of the Search Space

As presented in Section IV, Heuristics Miner++ parameters are not independent. These dependencies can be characterized by listing the main operations performed by the mining algorithm and the corresponding parameters (as defined in Section II-A):

1) calculation of the length-one loop and check of parameters (d) and (b);
2) calculation of the length-two loop and check of parameters (e) and (b);
3) calculation of the dependency measure and check of parameters (a), (b) and (c);
4) calculation of AND measure and check of parameter (g);
5) calculation of long distance measure and check of parameter (f).

When more than one parameter is checked in the same operation, all the corresponding checks have to be considered as in 'and' relation, meaning that all constraints must be satisfied. The most frequent parameter that is checked is the (b) (positive observation threshold), occurring in three steps; under these conditions, if, as an example, the dependency relation under consideration does not reach a sufficient number of observations in the log, then the check of parameters (a), (c), (d) and (e) can be skipped because the whole check (the 'and' with all other parameters) will not pass, regardless of the success of the single checks involving the (a), (c), (d) and (e) parameters.

Besides that, there are some other rules intrinsic on the design of the algorithm: the first is that if an activity is detected as part of a length-one loop, then it can't be in a length-two loop and vice versa (so, checks in step 1) and step 2) are in mutual exclusion); another is that if an activity has less than two exiting edges then it is impossible to have an AND or XOR split (and, in this case, step 4) does not need to be performed).

In order to simplify the analysis of the possible mined networks, we think is useful to distinguish two types of networks, based on the structural elements that compose them:

- "Simple networks", which includes process models with no loops and no long distance dependencies;
- "Complete networks", which includes simple networks extended with at least one loop and/or one long distance dependency.

For the creation of the first type of networks, only steps 3) and 4) (on the list at the beginning of this section) are involved, and so only parameters (a), (b), (c) and (g) have an important role in the creation of this class of networks. Complete networks are obtained by adding, to a simple network, one or more loops, by using steps 1) and 2), and/or one or more long distance dependencies via step 5). It can be observed that, once the value for parameter (b) is fixed, steps 1), 2), and 5), are in practice controlled independently by parameters (d), (e), and (f), respectively.

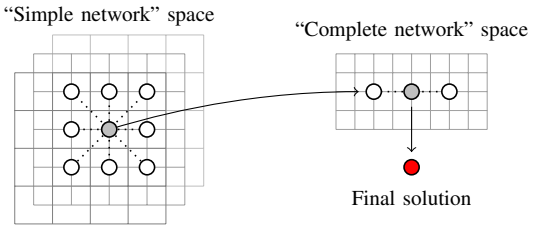

Fig. 2. A graphical representation of the searching procedure: *(left hand side)* the system looks for the best solution within the parameters subspace generating the 'simple network' class of process models. When a (local) optimal solution is found, the system tries to improve it by moving into the 'complete network' space *(right hand side)*.

In the following, we explain how we have exploited this factorization to perform the search for the "best" process model.

B. Searching for the Best Hypothesis

At this point, the new objective is the definition of the process for the identification of the "best" model (actually, we have to find the best parameters' configuration). There are two issues here: the first is the definition of some criterion to define what means "best model". Secondly, there is the problem of the hypothesis space that is too big to be exhaustively explored. Let start from the latter problem, assuming that we have a criterion to quantify the goodness of a process model.

For what concerns the big dimension of the search space, we start the search within the class of simple networks and, once the system finds the "best" local model it tries to extend it into the complete network space. With this division of the work the system reduces the dimensionality of the search spaces.

From an implementing point of view, in the first phase, the system has to inspect the joint space composed only of the parameters (a), (b), (c) and (g) (the parameters involved in "simple networks") and, when it finds a (potentially only local) optimal solution, it can try to extend it introducing loops and long dependency. In Fig. 2 we propose a graphical representation of the main phases of the exploration strategy. Of course, this search strategy is not complete for two reasons: *i)* local search is, by definition, not complete; and *ii)* the "best" process model may be obtained by extending with loops and/or long dependencies a sub-optimal simple network.

In order to complete the definition of our search strategy, it remains to give a formal definition of our measure of "goodness" for a process model. To this aim, we adopt the Minimum Description Length (MDL) principle [6]. MDL is a widely known approach, based on the Occam's Razor: "choose a model that trades-off goodness-of-fit on the observed data with 'complexity' or 'richness' of the model". Let's take as an example the problem of communicating through a very expensive channel: we can build a compression algorithm whereby the most frequent words are represented in the shortest way and, the less frequent have a longer representation. Now, as first thing to do, we have to transmit the algorithm and then we can use it to send our encoded messages. We have to pay attention in not building a too complex (that can handle many cases) algorithm: its transmission may neutralize the benefits of its use, in terms of total amount of data to be transmitted. Consider now the set H of all possible algorithms that can be built and, given $h \in H$, let $L(h)$ be its description size and $L(D|h)$ will be the size of the message D after its compression using h. The MDL principle tells us to choose the 'best' hypothesis h_{MDL} as

$$h_{MDL} = \underset{h \in H}{\operatorname{argmin}}\ L(h) + L(D|h).$$

The same situation can be reproduced in our context: the algorithm size becomes the 'complexity' of the mined model and the message size becomes the number of log traces that the mined model can 'explain'.

In this work we studied, as model complexity, the number of edges in the network. This is an easily computable measure, although it may underestimate the complexity of the network because it disregards the different constructs that compose the network. Anyway, this is a good way characterize the description length of the process model.

As $L(D|h)$ measure, we use the fitness measure introduced in [8] and, in particular, we opted for the *continuous semantics* one. Differently from the *stop semantics*, the one chosen does not stop at the first error, but continues until it reaches the end of the model. This choice is consistent with our objective to evaluate the whole process model. This measure is expressed as:

$$f_{M,W} = 0.4 \cdot \frac{parsedActs(M,W)}{|\mathcal{A}_W|} + 0.6 \cdot \frac{parsedTraces(M,W)}{logTraces(W)}$$

where M is the current model and W, as usual, is the log to "validate"; $|\mathcal{A}_W|$ is the number of activities in the log and $logTraces(W)$ is the number of traces in W; $parsedActs(M,W)$ gives the sum of all parsed activities for all traces in W and $parsedTraces(M,W)$ returns the number of traces in W completely parsed by the model M (when the final marking involves only the last activity).

The search algorithm starts from a random point in the "simple network" space and, following a hill-climbing approach, evaluates all the neighbour simple networks obtained by moving the current value of one of the parameters up or down of a position within the discretized space of possible values. If there exists a neighbour network with a better MDL value, then that network becomes the current one and the search is resumed until no better network is discovered. The "optimal" simple network is then used as starting point for a similar search in the remaining parameters space, so to discover the "optimal" complete network, if any.

In order to improve the quality of the result, the system restarts the search from another random point in the hypothesis space. At the end, only the best solution among all the

ones obtained by the restarts is proposed as "final optimal" solution.

VI. Experimental Results

In order to evaluate our approach we tried to test it against a large dataset of processes. In order to assign a score to each mining, we built some random processes and we generated some logs from these models; starting from these the system tries to mine the models. Finally, we compared the original models versus the mined ones.

A. Experimental set up

The set of processes to test is composed of 125 process models. These processes were created with a tool that we developed for the random generation of this kind of processes: Process Log Generator[3].

The generation of the random processes is based on some basic "process patterns", like the AND-split/join, XOR-split/join, the sequence of two activities, and so on. In Figure 3 some statistical features of the dataset are shown. For each of the 125 process models, two logs were generated: one with 250 traces and one with 500 traces. In these logs, the 75% of the activities are expressed as time intervals (the other ones are instantaneous) and 5% of the traces are noise. In this context "noise" is considered either a swap between two activities or removal of an activity.

We tried the same procedure under various configurations: using 5, 10, 25 and 50 restarts. In the implemented experiments, we run the algorithm allowing 0, 1, 5 and 10 lateral step, in case of local minimum (in order to avoid problems in case of very small plateau).

The distance of the mined process from the correct one is evaluated with the F1 measure, which is the harmonic mean between precision and recall: 'true positives' are the correctly mined dependencies between activities, 'false positives' are dependencies present in the original model but not in the mined one, and 'false negatives' are dependencies present in the mined model but not in the original one.

B. Results

The number of improvement steps performed by the algorithm is reported in Fig. 4. As shown in the figure, if the algorithm is run with no lateral steps, then it stops early. Instead, if lateral steps are allowed, the algorithm seems to be able, at least in some cases, to get out of plateaus. In our case, even 1 step shows a good improvement in the search. The lower number of improvement steps (plot on the right hand side), in the case of 500 traces, is due to the fact, with more cases, it is easier to reach an optional solution.

The quality of the search mining result, as measured by the F1 measure, is shown in Fig. 5. Results for 250 traces are reported in the left hand side plot, while results for 500 traces are shown in the right hand side plot. It is noticeable that the average F1 is higher in the 500-traces case.

[3]See the project web page, at `http://www.processmining.it/sw/plg`, for more information.

VII. Conclusions and Future Work

In this paper we have discussed an important problem that prevents the wide use of process mining algorithms able to deal with noisy logs, i.e. how to set the mining parameters. We focused on a specific process mining algorithm, i.e. Heuristics Miner++, and proposed to discretize the parameters space according to the traces in the log. Then we suggested to perform a constrained local search in that space to cope with the complexity of exploring the full set of candidate process models. The local search is driven by the MDL principle to look for the process model trading-off the complexity of its description with the number of traces that can be explained by the model. The experimental results obtained on a set of randomly generated processes seem to be encouraging.

In principle, the same approach can be applied to other similar mining procedures.

Possible improvements of the technique can involve the increase in the number of explored hypothesis. A dynamic generation of the hypothesis space could help to cope with the corresponding computational burden. Another improvement that we think can be very useful is the introduction of machine learning techniques, that may allow the system to learn which process patterns have to be preferred in the search and to improve the "goodness measure" by directly encoding this information.

Acknowledgment

Andrea Burattin's work was supported by SIAV S.p.A.

References

[1] W. M. P. van der Aalst, "Process Discovery: Capturing the Invisible," *IEEE Computational Intelligence Magazine*, vol. 5, no. 1, pp. 28–41, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5386178

[2] W. M. P. van Der Aalst and A. J. M. M. Weijters, "Rediscovering workflow models from event-based data using Little Thumb," *Integrated Computer-Aided Engineering*, vol. 10, no. 2, pp. 151–162, 2003.

[3] W. M. P. van der Aalst and A. J. M. M. Weijters, *Process-Aware Information Systems: Bridging People and Software through Process Technology*. John Wiley & Sons Inc, 2004, ch. 10: Proces, pp. 235–255.

[4] A. Burattin and A. Sperduti, "Heuristics Miner for Time Intervals," in *ESANN*, Bruges, Belgium, 2010.

[5] W. M. P. van der Aalst, C. W. Günther, V. Rubin, H. M. W. Verbeek, B. F. Dongen, and E. Kindler, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software & Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2008. [Online]. Available: http://www.springerlink.com/index/10.1007/s10270-008-0106-z

[6] P. Grünwald, *A tutorial introduction to the minimum description length principle*, P. Grünwald, I. J. Myung, and M. Pitt, Eds. MIT Press, 2005.

[7] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, "The ProM framework: A new era in process mining tool support," *Application and Theory of Petri Nets*, vol. 3536, pp. 444–454, 2005.

[8] W. M. P. van Der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters, "Using Genetic Algorithms to Mine Process Models: Representation, Operators and Results," *BETA Working Paper Series, WP 124, Eindhoven University of Technology*, 2004.

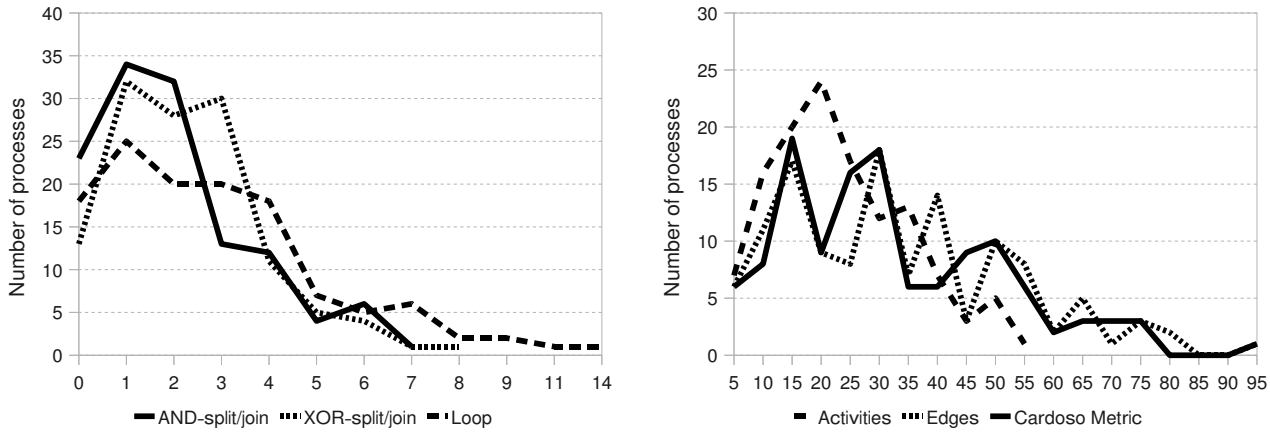

Fig. 3. Main features of the processes dataset built. The first plot, at the left hand side, reports how many processes (y-axis) contain a given number of AND-split/join (x-axis); a similar curve is reported for XOR-split/join and for the number of loop. The plot in the right hand side contains the same distribution versus the number of edges, the number of activities and the Cardoso metric (all these are grouped using bins of size 5).

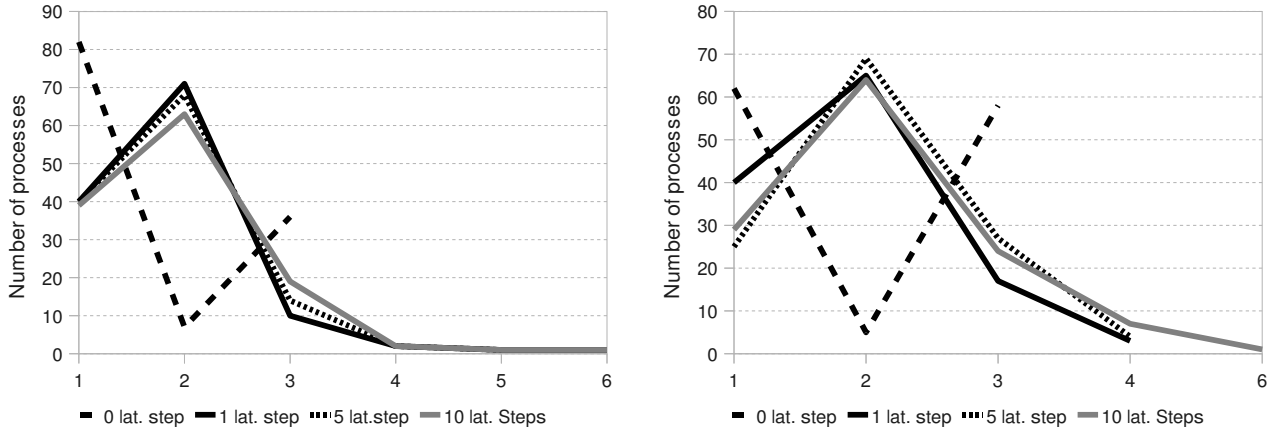

Fig. 4. Number of processes whose best hypothesis is obtained with the given number of steps, under the two conditions of the mining (with 0, 1, 5 and 10 lateral steps). The left hand side plot refers to the processes mined with 250 traces while the right hand side refers to the mining using 500 traces. Both these datasets consist of 125 processes and have been generated with a 5% of noise and the 75% expressed as time intervals (the other ones are instantaneous).

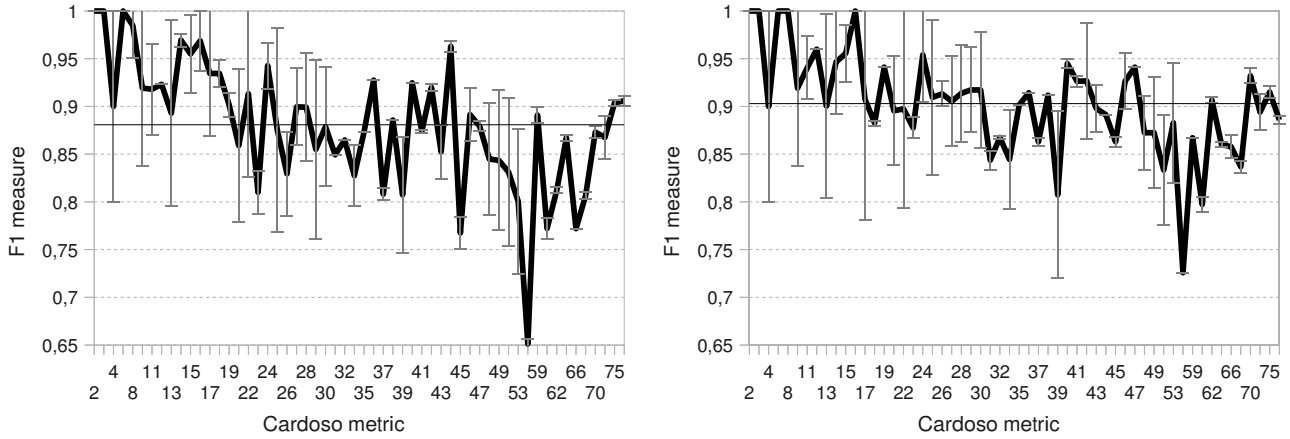

Fig. 5. "Goodness" of the mined networks, as measured by the F1 measure, versus the size of the process (in terms of Cardoso metric). The left hand size plot refers to the mining with 250 traces, while the right hand side plot refers to the mining with 500 traces. Both these datasets consist of 125 processes and have been generated with a 5% of noise and the 75% expressed as time intervals (the other ones are instantaneous).