Abstract

This document is devoted to discuss some issues concerning the complexity and the implementation of the proposed algorithms (Section A), and further results of experiments on real data (Section B).

A Details on Complexity and Implementation Issues

A.1 Algorithm OASC

We next provide a complexity analysis of algorithm OASC, assuming that the input log consists of \( N \) traces, spanning across \( T \) tasks, and that the maximal size of the \( S \)-patterns used for the clustering is \( S \) (i.e. \( \text{pattSize} = S \)). Let us first focus on function FindPatterns, while denoting by \( P_{\text{can}} \) the maximum number of patterns generated at any iteration of its main loop (Steps P3-P9).

To keep things simple, let us assume that each log trace contains at most one occurrence (or a constant number of occurrences) of any task, i.e., the unknown workflow process that generated the log is acyclic. It is worth noticing that this assumption lowly impacts on our cost analysis, and that usually the number of iterations of any task cycle is bounded in real-world processes. On the other hand, some preprocessing techniques and tools (e.g., in ProM [1]) have been developed to eliminate duplicated activities from log traces, in order to make simpler process mining analyses.

Moreover, we assume that the set \( L_1 \) is stored in a sparse matrix of size \( T^2 \), which can be computed in \( O(N \times T) \) time (Step P1), provided a reasonably high (e.g., \( \sigma > 0.04 \)) support threshold is used\(^1\). In fact, search-tree structures can be used to keep

---

\(^1\)By putting \( \sigma \) far from 0, the number of spurious tasks admitted between the vertices of any elementary (single-edge) pattern in \( L_1 \) gets lower than \( T - 1 \). Indeed, recall we are assuming that log traces are long \( T \) at most. For example, for \( \sigma = 0.05 \) the maximum number of spurious activities is \( 2 \approx \lceil -\ln(0.05) \rceil \).
the patterns generated in the two latter iterations of the loop over Steps P3-P9, and to efficiently prevent the insertion of duplicated candidates\(^2\). Under this assumption, generating any higher level of candidate patterns (Step P4) only consumes \(O(T \times P_{can})\) time. Calculating the support scores of level-\(i\) candidate patterns (Step P5) can be done instead in \(O(N \times (T + P_{can} \times i))\)—the support of all edges contained in each candidate pattern is evaluated for each log trace \(t\) encountered in the current scan of the log, after suitably converting \(t\) into a vector storing the position of all the activities in \(t\).

Therefore, the overall computation time of function \(\text{FindPatterns}\) is \(O(N \times (T + P_{can} \times S^2) + T \times P_{can})\), with the rightmost addend playing no relevant role in most practical cases. Clearly, this cost is influenced by parameter \(P_{can}\), which may be combinatorial in the number \(T\) of process tasks. In fact, this problem is inherent to the levelwise generation-and-test strategy adopted by the function, which risks producing a great deal of candidate patterns, similarly to several classical pattern mining algorithms, such as \text{Apriori}, \text{AprioriAll} and \text{GSP} (see, e.g., [3]). A pessimistic estimate for the number of level-\(i\) \(S\)-patterns is indeed \(|L_i| = O(T \times (T^{-1}))\), which actually corresponds to the rare case of a completely chaotic process logs, featuring most of all possible task sequences, combined with the usage of very low values for the minimum support threshold \(\sigma\). In actual fact, \(P_{can}\) also strongly depends on \(\sigma\) and on the actual distribution of activity sequences in the given log. In typical situations, where a more aggressive pruning strategy is used and the execution of process activities obeys stricter precedence constraints and routing rules, the actual number of patterns at each step of the levelwise computation is far lower than the above estimate, and not really combinatorial in \(T\).

The complexity of function \(\text{FindCoClusters}\) mainly depends on the number \(P_{max}\) of maximal patterns eventually used for the clustering—notice that the maximal patterns returned by \(\text{FindPatterns}\) are usually far less than \(P_{can} \times S\), and that the analyst can anyway decide to only keep the a restricted number of them having the top-\(P_{max}\) support scores. Let us assume that threshold \(\alpha\) is chosen in a way that allows to identify, for any trace \(t\), any pattern \(p\) with \(\text{supp}(p,t) > \alpha\) by only using a sliding window over the activities of \(t\). In this case, the similarity matrix over the patterns can be computed in linear time (i.e. in \(O(N \times (T + P_{max} \times S))\)) with respect to the size of the log (cf. Step C1). Applying algorithm \text{MCL} [2] for a preliminary clustering of these \(P_{max}\) patterns (cf. Step C2) takes \(O(P_{max} \times K^2)\), where \(K\) is a parameter controlling matrix sparsity (precisely, the number of nonzero entries per column) in the computation of matrix squares, which is typically set between 500 and 1500. Step C5 in the loop requires to evaluate the support of all patterns w.r.t. each log trace, which can still be done in \(O(T + P_{max} \times S)\). Assuming that the set \(P\) of pattern clusters is stored search-tree-based structure, the following Steps C6-C11 do not augment the complexity, since they can

\(^2\)We actually represent any size-\(i\) (with \(i > 1\)) fork (resp. join) pattern as a pair consisting of: (i) the synchronization (resp. splitting) task \(a\) shared by all of its constituent edges, and (ii) a bitmap vector encoding the set of tasks that follow (resp. precede) \(a\) in the pattern. At iteration \(i\), all the patterns with size equal to \(i - 1\) or \(i\) and hinged in the same task are stored in a separate search tree, and kept ordered with respect to their respective bitmap elements.
be done in linear time w.r.t. the number of patterns \( P_{\text{max}} \), which is indeed logarithmic in the maximum number of pattern clusters. Therefore, the overall cost of function \( \text{FindCoClusters} \) is \( O(N \times (T + P_{\text{can}} \times S) + P_{\text{max}} \times K^2) \).

In conclusion, since \( P_{\text{max}} < P_{\text{can}} \times S \), the overall complexity of algorithm \( \text{OASC} \) is:

\[
O(N \times (T + P_{\text{can}} \times S^2) + T \times P_{\text{can}} + P_{\text{max}} \times K^2)
\]

Notice that the rightmost addend in the above formula can be usually disregarded for practical application cases.

Let us finally observe that the algorithm requires a limited number of scans over the input log, and can be implemented without importing the input log as a whole into the main memory. Indeed, the input log can just be scanned \( S \) times (\( S < 10 \) works fine on typical process logs) for finding patterns of size \( S \), plus two further times for building matrix \( M \) and for assigning each trace to the various clusters (Steps C4-C12). Thus, main memory computation is just limited to the clustering of the interesting patterns (whose number is generally small compared with the input log—in any case, one usually desires to focus on the most frequent ones). This property guarantees potential scaling over huge datasets.

A.2 Algorithm \text{LearnDADT}

Let us now study the complexity of \text{LearnDADT}, for a log consisting of \( N \) traces, and associated with \( T \) activities and \( F \) attributes. As a reasonable worst-case situation, we can think of a “complete” decision tree built over continuous attributes, where each leaf just contains one trace and the number of levels is \( H = O(\log(N)) \). Notice that when all the attributes are discrete (or have been discretized preliminary), \( H = O(\min(F, \log(N))) \), for any root-to-leaf path can contain each of them at most once. In fact, we are implicitly assuming here, as done in [5], that the attributes provide enough tests to precisely differentiate the training instances, and to always split them in a balanced way.

In such a case, when creating the nodes of level \( i \), all the \( N \) log traces must be scanned in order to compute the binary relation \( \prec^l_{\sigma'} \) over the activity pairs, for each node and for all its associated active clusters. Regarding the number of clusters as a constant for the sake of simplicity, this will take \( O(N \times T^2) \) computation time. This scan over all the log traces also suffice to compute, for all level-\( i \) nodes, the split scores (see line B3) for, at most, all the \( F \) attributes (in the worst case where they are all continuous).

Carrying out such a computation for all the tree levels requires \( O(N \times H \times F) \) steps. This latter upper bound holds as well for the cost of checking temporal compliance (Step B5) over all the nodes of the tree\(^3\).

Therefore, we can conclude that the overall computational cost of \text{LearnDADT} is:

\[
O(N \times (H \times F + T^2))
\]

\(^3\)For each level-\( i \) node, say \( v \), the compliance check only needs to look up the \( \prec^l_{\sigma'} \) relation already computed for \( v \), relatively to the split attribute chosen for it and to the (at most \( F \)) attributes occurring in the path from the root to \( v \)’ parent. Since, in the worst case, the tree contains \( H \) levels, each consisting of at most \( N \) nodes, no more than \( O(N \times H \times F) \) time is to be spent.
Notice that this estimate also encompasses other additional costs that have not taken into account above: preliminary sorting the values of all continuous attributes (taking \(O(F \times N \times \log N)\)), in order to efficiently identifying their optimal split points in the tree growth process, and pruning the tree according to the subtree replacement method \([5]\) (which takes \(O(N)\)).

As mentioned in the paper, since in most cases the tree built in the growing phase is quite rarely "complete" (i.e. \(H << O(\log(N))\)), algorithm LearnDADT is usually far less time-consuming than as above described — often linear time in \(N, A, F\) is sufficient. Moreover, in order to efficiently work with large-scale logs and speed up the execution time, we are investigating the possibility of combining our approach with external-memory and parallelized DT-induction ones (see \([4]\) for detailed references)—some of which (e.g. RainForest) naturally combine with our C4.5-like scheme.

A.3 Scalability Tests on Synthesized Process Logs

In the experiments described below, we tried to complement our theoretical study of our approach’s complexity with an empirical analysis of its efficiency and scalability. Owing to its potentially critical computational costs (linked to the potential combinatorial explosion of candidate pattern sets), we here only focus on some tests performed with algorithm OASC on some synthesized logs, generated according to the procedure described in the paper. In particular, we created a number of datasets with increasingly larger number of traces and activities in the schema, while fixing \(p^{\text{min}}=0.02\), \(N_A=125\), and \(N_C=4\). Fixed values were taken as well for all OASC’s parameters: \(\alpha=0.2\), \(\beta=0.5\), \(\gamma=4\) and \(\sigma=0.15\), and \(\text{pattSize}=10\). As shown in Figure 1.(a), the total computation time linearly scales both with the number of log traces \(N_T\) and with the size of the process schema \(W_P\) used to generate the log itself (i.e. \(S_P\)).

Finally, Figure 1.(b) reports the total computation time spent by algorithm OASC, when keeping fixed all the parameters but \(\sigma\) and \(\gamma\) (actually, we set again \(\alpha=0.8\), \(\gamma=4\), \(\beta=0.5\) and \(\text{pattSize}=8\)). Note that \(\sigma\) and \(\gamma\) do impact on computation time: the lower their value the higher the time. However, a notable increase only occurs when \(\sigma\) passes
from 0.1 to 0.05. This effect is emphasized when γ too is kept low, and any σ-frequent pattern is also (σ, γ)-maximal.

B Detailed Results of Tests on Real Process Logs

B.1 Tests on Log A

Each trace of Log A encodes the sequence of basic operations (i.e., MOV, DRB, DRG, LOAD, DIS, SHF, OUT) applied to each single container. In addition to container attributes, a series of data attributes were associated with each occurrence of these operations, including the human that carried out the operation (Originator), the two positions the container was moved between (FromPosition and ToPosition, resp.), the completion time (Timestamp) and the duration of the operation (ElapsedTime), the distance covered (Distance) and the kind of vehicle used in the operation (Vehicle).

As an example, we next discuss the results obtained when applying our approach to the log described above with \( \omega = 0.6 \), considering all data attributes for DADT induction. In this case, algorithm OASC discovered two distinct normal usage scenarios, and 53 outlier traces. Subsequent analyses by domain experts confirmed that most of outlier individuals (i.e., container histories) actually correspond to anomalous cases and to malfunctions in the tracking system. Structural aspects of the scenarios are described by the workflow schemas shown in Figure 2, which essentially differ for the presence of operations performed with multi-trailer vehicles: the schema of Figure 2.(a) does not feature any of these operations, which are instead contained in the other schema. Notably, the former schema captures the vast majority of handling cases (4736 containers of the original 5389 ones). This reflects a major aim of yard allocation strategies:

\[ \text{(a) Cluster 0} \quad \text{(b) Cluster 1} \]

Figure 2: Results on log A (\( \omega = 0.6 \) and \( \sigma = 0.1 \)): the two workflow schemas found.
to keep each container as near as possible to its positions of disembarkation/embark, by performing short transfers via straddle-carriers.

Interestingly, high quality scores were obtained by these structural models over all of the conformance measures: $Fitness = 0.8791$ and $BehAppr = 0.9089$ (cf. Table 1 in the paper). Moreover, an astonishing 97.49% accuracy score (cf. Table 1 in the paper) was achieved by the data-aware classification model discovered for the log, so confirming that these two markedly different execution scenarios strongly depend on process features that go beyond the mere sequencing of yard operations.

Based on Table 1 — which reports 4 top-ranked attributes for each dataset, i.e. the 4 attributes that most frequently appeared in the the top levels of the decision trees discovered from each dataset — we can see that the following container properties stood out among these features: the provenance port of a container ($PrevHarbor$), the kind of ship that is going to take it away ($ShipType_{OUT}$), the navigation line delivering the container to the hub ($NavLine_{IN}$) and the kind of container ($ContType$) (e.g., fridge container). As to the attributes of tasks (i.e., operations), we notice that only those associated with the $MOV$ operation are actually used by the classification model.

In Figure 3, we report two interesting branches of the decision tree induced for this log, which were deemed quite useful for explaining and discriminating the discovered usage scenarios. Both branches are represented as if-then rules (i.e., $r_1$ and $r_2$, respectively). Notably, $r_1$ is a very selective rule classifying just 11 traces of the overall 5336 as belonging to Cluster 0 with an accuracy of 81.82%. Conversely, rule $r_2$ assigns 52 traces of the input log to Cluster 1 and gets 94.23% accuracy.

A finer grain analysis, conducted with the help of Table 2 (where individual precision/recall measures for the two clusters are shown), confirms that the model guarantees a high rate of correct predictions for either cluster.

**B.2 Tests on Log B**

We recall that this log was created to arrange original data into a “position-centric” fashion, in order to capture the paths typically followed by the containers around the yard. Precisely, each trace in Log B encodes the sequence of yard sectors occupied by

```
if PrevHarbor = ANR 
and ShipType_{OUT} ≠ RR 
and ShipType_{OUT} ≠ CF 
and ContType = DC 
then MOV::Distance ≤ 204

r_1: if PrevHarbor = ANR 
and ShipType_{OUT} ≠ RR 
and ShipType_{OUT} ≠ CF 
and ContType = DC 
then Cluster 0

r_2: if PrevHarbor = ANR 
and ShipType_{OUT} ≠ RR 
and ShipType_{OUT} ≠ CF 
and ContType = DC 
then 204 < MOV::Distance ≤ 354 
then Cluster 1
```

Figure 3: Results on log A: an excerpt of the decision tree.

<table>
<thead>
<tr>
<th>Test Dataset</th>
<th>Attributes</th>
<th>A@0.6</th>
<th>Top Level Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log A</td>
<td>case+task</td>
<td>0.6</td>
<td>$PrevHarbor, ShipType_{OUT}, NavLine_{IN}, ContType$</td>
</tr>
<tr>
<td>Log B</td>
<td>case+task</td>
<td>0.6</td>
<td>$ShipSize_{IN}, ShipType_{OUT}, ContType, PrevHarbor$</td>
</tr>
<tr>
<td>Log CAD</td>
<td>task</td>
<td>0.6</td>
<td>$Creation::Group, Creation::Role, Construction::Group, Share::Group$</td>
</tr>
</tbody>
</table>

Table 1: Top level attributes in the DADT models.
a single container during its stay. Each log event is also associated with several non-
structural data attributes, which include the human who moved the container (Origin-
ator), the distance covered (Distance), the time spent to move the container (Elapsed-
Time), the kind of vehicle used (Vehicle), and the working turn during which it hap-
pened (Turn). We next focus on the results obtained in one of the tests we carried out
on this process log, where the parameter \( w \) was set again to 0.6, and all data attributes
were taken into account for inducing the DADT model.

The structural clustering performed by algorithm OASC allowed to recognize 5 trace
clusters, corresponding to prevalent behaviors, and 63 outlier traces. We remark that,
in principle, due to the high number of sectors and moving patterns that come to play
in such analysis perspective, any flat representation of container flows, just consisting
of a single workflow schema, risks being either inaccurate or difficult to inter-
pret. Conversely, by separating different behavioral classes our approach ensures a
modular representation, which can better support explorative analyses. In fact, the
five clusters found in this test have been equipped with clear and compact workflow
schemas, which exhibited high levels of conformance with the log: Fitness = 0.8687
and BehApp = 0.9254 (cf. Table 1 in the paper). As instance, two of these schemas are
shown in detail in Figure 4, which differ both in the usage of sectors and in some of the
paths followed by the containers across these sectors.

Interestingly, a satisfactory accuracy (95.01\%) is achieved again by the DADT
model. As a matter of fact, by comparing these results with those obtained in the

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Size</th>
<th>P</th>
<th>R</th>
<th>( P (b = 1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4736</td>
<td>98.24%</td>
<td>98.94%</td>
<td>98.59%</td>
</tr>
<tr>
<td>1</td>
<td>600</td>
<td>91.17%</td>
<td>86.00%</td>
<td>88.51%</td>
</tr>
</tbody>
</table>

Table 2: Results on log A (\( w = 0.6 \) and \( \sigma = 0.1 \)): details on the discovered clusters.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Size</th>
<th>P</th>
<th>R</th>
<th>( P (b = 1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1600</td>
<td>94.29%</td>
<td>97.06%</td>
<td>95.65%</td>
</tr>
<tr>
<td>1</td>
<td>1068</td>
<td>88.99%</td>
<td>92.15%</td>
<td>90.54%</td>
</tr>
<tr>
<td>2</td>
<td>344</td>
<td>95.00%</td>
<td>93.14%</td>
<td>94.06%</td>
</tr>
<tr>
<td>3</td>
<td>187</td>
<td>94.57%</td>
<td>64.89%</td>
<td>76.97%</td>
</tr>
</tbody>
</table>

Table 3: Results on log B (\( w = 0.6 \) and \( \sigma = 0.1 \)): details on the discovered clusters.
previous test, we notice a slightly lower precision and a larger size of the decision tree (cf. Table 3 in the paper), mainly due to the higher level of complexity that distinguish the position-centric analysis from the operation-centric one.

Incidentally, Table 3 reveals that such worsening is mainly to blame on the inability of $DADT$ to appropriately recognize well the Cluster 1, which is, in fact, slightly confused with the Cluster 3.

We finally notice that almost the same attributes as in the former test have been employed to discriminate the clusters (even though in a different order), except for the usage of $ShipSize_{IN}$ (i.e., the size category of the ship that delivered the container) in place of $NavLine_{IN}$ (cf. Table 1).

**B.3 Tests on Log CAD**

Let us now focus on one experiment conducted over this log, where the proposed approach was used with $\alpha = 0.6$ and by considering all data attributes. Four different clusters and associated workflow models were discovered in this experiment, which capture the behavior registered in the log in an adequate enough manner—the global conformance scores of the structural model are, indeed, $Fitness = 0.6933$ and $BehApp = 0.6687$. In fact, 50 traces were perceived as outliers, which actually corresponds to unusual developments of CAD projects.

A $DADT$ model was also discovered, consisting of 45 nodes, which achieves 72.47% prediction accuracy, based on the information about the role and group of the users that performed some of the CAD operations—primarily, Creation, Construction and
\textbf{Figure 6:} Results on log \textit{CAD} ($\omega = 0.6$ and $\sigma = 0.1$): an excerpt of the decision tree.

\textbf{Share.} Despite a lower precision score is achieved than in the previous application scenario, this result is quite surprising, as there was no a-priori expectation that users’ roles and groups could be correlated with different CAD scenarios, and could really help discriminate among them. Figure 5 shows the workflow models discovered for two (of four) clusters found. In particular, it is easy to see that \textit{Cluster 3} corresponds to a somewhat anomalous execution case—which actually regards only 13 traces—where a project is built but it is never validated. Interestingly, the DADT classification model discovered in the experiments is able to precisely predict even this outsider behavior (see Table 4).

In Figure 6, we show four rules extracted from DADT model, all of which are quite interesting and accurate. Specifically, \textit{r3} is a simple and yet very precise two-level rule which only captures 6 traces of \textit{Cluster 1}, with 100\% accuracy. This is interesting if we consider the disappointing performances the tree has in predicting this cluster (cf. Table 4). Instead, 592 traces of the input log are assigned to \textit{Cluster 0} by rule \textit{r4}, which also achieves maximal precision (100\% accuracy). Rule \textit{r5} gets some lower precision result—it classifies 339 traces with 66.05\% of accuracy— but it is still helpful, in that it evidences the compliance of the tree w.r.t. to schemas shown in Figure 5. Finally, rule \textit{r5}, which correctly assigns 11 log traces to \textit{Cluster 0}, demonstrates that satisfactory results (an accuracy of 92.46\%) can be achieved despite of the constraints imposed by precedence relationships of the workflow schemas.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{Cluster} & \textbf{Size} & \textbf{P} & \textbf{R} & $F_{\beta=1}$ \\
\hline
0 & 3734 & 78.13\% & 87.85\% & 82.71\% \\
1 & 825 & 56.99\% & 19.33\% & 29.32\% \\
2 & 1138 & 56.08\% & 92.38\% & 67.82\% \\
\hline
\end{tabular}
\caption{Results on log \textit{CAD} ($\omega = 0.6$ and $\sigma = 0.1$): details on the discovered clusters.}
\end{table}

\textsuperscript{4}Notice that original German names appear for some of the tasks: \textit{Musterbau, Pruefung, TechAend} and \textit{NullSerie}, which correspond to the tasks \textit{Prototyping, Test, TechRevision} and \textit{PilotSeries}, respectively.
Bibliography


