Learning agent organisations: studies on collaborative modelling, performance management and learning capacities of networks of collaborative agents
Mulder, W.

Citation for published version (APA):
3 COLLABORATIVE GRAMMAR MODELLING

We present a methodology for learning structures of successive events that occur in multi domain agent organisations. First we explain the setting and mechanism, then we describe how we learn DFA models from network event-data. We explain how MDL is used and show how DFA models learned in single domains can be merged.

3.1 Learning software agents

3.1.1 L-agents and N-agents

In this chapter we look at software agents that collaborate in a learning task. Our goal is to design a distributed learning mechanism that is able to learn structures in distributed datasets. To stay close to the main subject of the dissertation, one can think of the datasets to contain information about events, for example events that occur in a collaborative agent organisation that handles tasks. For simplicity we make a distinction between agents that observe events and learn structures and agents that handle operational tasks. The two types of agents are labelled as N-agents (network agents) and L-agents (learning agents). Note that the L-agents have a (learning)task which is not to be confused with the (generic) tasks of the N-agents.

The distinction between L- and N-agents allows us to explain our collaborative learning mechanism in which learning is considered as a task. We will in a later stage leave this explicit distinction (in chapter 6), where we consider learning more as a process than as a task, and regard learning in organisations more from a perspective of self-organisation. In this chapter, the L-agents will get implemented as software agents while the N-agents just represent the entities that belong to the organisation (network) of which we want to optimise the task-handling behaviour. The L-agents will implement our collaborative modelling mechanism, described in the second half of the chapter.
Using such a perspective one may talk about the management of one network carried out by the other. The task of the L-agents network is to model the behaviour of the N-agents network. The N-agents may use the models of the L-agents. The overall aim is to let the N-agents better understand the global behaviour of the network, and enable them to tune their own behaviour such as to improve the performance of the network as a whole.

Since both types of networks interact which each other they can be regarded as one single network. Using the perspective of self-organisation, the whole can be seen as one single network that learns by itself. Agents in such a network influence each other in order to optimise the network.

Figure 3.1 shows the two different agent organisations. The learning agents are denoted with an $L$, the operational network agents are denoted with a $N$. The agents operate in a universe $U$. The grey oval-shaped platform indicates the environment of the N-agents, which consists of multiple organisational domains (represented by dotted ovals).

![Figure 3.1: A collaborative organisation of learning agents (L-agents) modelling the task-handling behaviour of an organisation of other agents (N-agents). The interaction between the two organisations has to be understood in terms of information about events and communication of detected structures.](image)

An L-agent can observe events that occur in one particular domain of the network of N-agents only. Communication constraints withhold an L-agent to share detailed data across the organisational domain border. To be able to obtain models of the behaviour of the global network of N-agents, the L-agents are allowed to share their own local models. The collaborative task of the L-agents is to contribute to the optimisation of performance of the N-agents. Communication within the network of
L-agents is about their collaboration in their global modelling task. Communication inside the network of N-agents is solely about its originally intended operation to act in an environment U. Communication between the N-agents and L-agents is twofold: an N-agent provides an L-agent with event data and vice versa, and an L-agent provides an N-agent with event models. An L-agent that learns carries out three steps; it observes data, it creates models and it generalizes these models.

### 3.1.2 Learning local and global DFA models

The task of the L-agents is to model the flow of events in networks of N-agents. The obtained (DFA-)models represent the structure of events to occur in the networks of N-agents. The general idea is that using the models of the L-agents, the N-agents can improve the structure of their processes and thereby improve their performance. Each L-agent observes event-data within its own domain and uses grammar induction to build local models of that data. The agents are allowed to share these models across the domains in order to obtain global, multi-domain models.

In a learning task we deal with a large distributed set of event-data that is produced by the network of N-agents. We call this dataset the *global dataset*. A set of learners (L-agents) has the collaborative task to model this dataset. Each learner can only observe a part of the global dataset. Such a part, called the *local dataset*, represents event-data inside the domain that is visible to the learner. Thus, the global dataset represents event-data of the whole networked environment and a local dataset represents event-data within a certain domain of that environment. Using its local dataset, each learner builds its own models, called its *local* or *individual* models. These models reflect the grammatical structures in the local dataset. The learners share and merge their local models into a global model. Note that ‘global’ here means ‘extended beyond local’. In the limit, ‘global’ covers the events from the ‘whole’ or ‘total’ organisation of N-agents.

When a particular learner communicates a model to another learner, this model is received and treated as an hypothesis by the receiving agent. Hypotheses are merged into the receiving agents’ global model. By means of merging models, each agent tries to obtain a global model of the events that occur in the organisation of N-agents. A side effect of this mechanism is that since each L-agent is allowed to make decisions on the order of accepting and merging of the hypotheses, the global model of one L-agent can be different from that of other L-agents. In this way, our mechanism allows the organisation to have different versions of a possible global model of task-handling events of the N-agents organisation. Our main motivation for not using a central blackboard is scalability. Taking into account the communication constraints in our intended application domain, a practical reason for using at least two models per agent is that a clear separation can be maintained between local information and shared information.
3.2 Collaborative modelling

3.2.1 Local DFA modelling

Based on time intervals, we collect event data. Events are labelled with symbols. Successive events are represented by means of strings, called sentences, of concatenated symbols. Together, a set of sentences form a sample.

Learning a DFA starts with a sample, consisting of a set of sentences that are supposed to be accepted (or rejected) by the target DFA. Then, generalization (learning) takes place by merging states. Decisions on merging are made using the principle of Minimum Description Length (MDL) (see section 2.4.2). The MDL score that we calculate for each DFA consists of two parts: the number of bits needed to encode the model and the number of bits needed to encode the sample, given the model (Adriaans & Jacobs [6], Mulder & Jacobs [102]). The MDL score is the sum of both parts. The lower the sum, the better the model is assumed to be.

We used the “Blue-Fringe” algorithm (Lang et al. [83]) to determine the merge candidates. This algorithm maintains a core of “red” states, which are states of the final DFA, and a list of “blue” states, which are the states that are considered for either to be promoted to “red” or to be merged with a red state. This set of “blue” states consists of all children of “red” states that are not themselves “red” states. The “blue” states are the heads of sub-trees of the PTA. The algorithm uses MDL to decide on the next step, which is either to promote a “blue” state to “red” or to merge a “blue” state with one of the “red” states.

Promoting a “blue” state to “red” does not alter the MDL score, since it does not change the DFA, and thus does not change the set of accepted sentences. Therefore, states are only promoted to “red” if all attempted merges result in a worse MDL score. To decide on a particular merging step, all possibilities are tried, the MDL score of each of the results is computed, and the best one is chosen, after which this whole process is repeated, until all states are “red” states.

![Diagram of DFA](image)

Figure 3.2: The ‘blue-fringe’ algorithm. The red states (labelled R) are the identified parts of the automaton. The blue states (labelled B) are the current candidates for merging.
DFA learning: merging states using 'Red-Blue' and MDL

Determine red-states and blue-states: Colour the start state red and its direct children blue

while count(blue-states) > 0
  for each blue-state
    for each red-state
      try merge blue state and red-state
      call calcMDL returning MDLscore
      remember the situation that has the lowest MDLscore
    endfor
  endfor
if MDLscore < MDL_current_dfa then
  merge blue state with the red-state
else
  promote first blue-state of the tree to a red-state
endif

determine new list of blue-states: color all direct children of red states blue
endwhile

Algorithm 3.1, individual DFA learning using the “Blue-fringe” algorithm and the principle of MDL for making decisions about merging states.
3.2.1.1 Calculation of the model code

The first part, the number of bits necessary for the model, \( L_{mc} \), is based on the work of (Adriaans & Jacobs [6]) and calculated as follows: Suppose we have a DFA with \( n \) states and a number of transitions (edges) over an alphabet \( \Sigma \) of \( |\Sigma| \) symbols. To deal in a simple manner with the type notification of a node, i.e. whether it is accepting, or not, we include a generic accepting state in this number \( n \). This gives us \( n \cdot |\Sigma| \) possible state transitions. As each of these transitions can have \( n \) possible destinations, there are \( n^{n \cdot |\Sigma|} \) possible DFAs with \( n \) states. There is considerable redundancy as for each permutation of state 2..n, the DFA is equivalent.

The size of the model code is therefore equal to \( n^{n \cdot |\Sigma|} / (n - 1)! \) and thus, the length of an index identifying one specific DFA is \( \log_2 \left( \frac{n^{n \cdot |\Sigma|}}{(n - 1)!} \right) \)

\[ l_{mc1} = \log_2 \left( \frac{n^{n \cdot |\Sigma|}}{(n - 1)!} \right) \]

Our experiences were that in situations in which a DFA contains a high variation in the number of edges per state, an encoding based on counting edges instead of node-transitions results in a smaller, i.e. better model code. For those DFAs we used

\[ l_{mc2} = \log_2 \left( \frac{(|\Sigma| + 1)^n \cdot (n|\Sigma|)^{n,edges}}{(n - 1)!} \right) \]

or via a bit-table of size \((n, |\Sigma|)\) that encodes whether or not a state-symbol combination exists, giving:

\[ l_{mc3} = n \cdot |\Sigma| + \log_2 \left( \frac{n^{edges}}{(n - 1)!} \right) \]

The practical approach we used is to calculate them all, and take the minimum.

\[ l_{mc} = \min(l_{mc1}, l_{mc2}, l_{mc3}) \]

3.2.1.2 Calculation of the data-to-model code

We define an index set for a DFA \( A \) as the set that associates a unique natural number with each string that is accepted by \( A \). The size of this index set determines the second part of the MDL score, i.e. the number of bits to code data given the model, \( L_{dmc} \).
Suppose $L$ is the maximum length of a sentence in the sample. Let $S$ be the number of sentences in the sample. Obviously, this set of sentences is a subset of the set of all sentences with length less than or equal to $L$.

Then we calculate the total number of sentences $N$ with length less than or equal to $L$ that are accepted by the model. This can be done in a trivial, brute force way by generating all possible sentences and verify whether they are accepted by the model, or in a more elegant way, by walking recursively back from the end states, through the model and count the number of ways a state can be reached for each particular length of a sentence.

Now, there are $p = \binom{N}{S}$ ways to select $S$ sentences from $N$ sentences. So, an integer with possible values between 0 and $p-1$ uniquely identifies the sample. Thus for the data-to-model code we need $\log_2(p)$ bits.

$$l_{dmc} = \log_2 \left( \frac{N!}{S!(N-S)!} \right) = \log_2 \left( \frac{N}{S} \right)$$

Note that we only estimate the size of the index set, we do not need to create the index set itself.

---

**Algorithm 3.2**

`Calculate MDL score`

Calculate the various model code sizes: $L_{mc1}$, $L_{mc2}$, $L_{mc3}$
Take minimum of these model codes: $L_{mc}$
Let $L$ = the max length of a sentence in the sample
Calculate $N$, the number of sentences with length $\leq L$ that are accepted by the model
Let $S$ = the number of sentences in the sample
Let $P = \binom{N}{S}$ number of ways to select $S$ sentences from $N$ sentences
size of data-to-model code: $l_{dmc} = \log_2(P)$
return score = $L_{mc} + l_{dmc}$

Algorithm 3.2, calculation of the MDL-score of a DFA.
3.2.2 Combining DFA models

In our situation of collaborative learning we have to obtain a global DFA model from a set of local DFA models. Since we have not seen any previous work on such an approach yet and since we want to apply a working mechanism in the case-environments of grid computing, some of the decisions are made for practical reasons.

Furthermore, decisions on how and when the individual local models are merged require knowledge about the workflow of the networked system that has to be modelled.

Since it is common to classify events to happen either sequential or in parallel to each other, we use two different merging methods:

- complementary DFA merging
- congruent DFA merging

Complementary DFA merging means that models from different environments are stitched (head-to-tail) into a single one.

Congruent DFA merging means that models are merged by means overlapping their start-states together with the rest of their states.

After merging, either complementary or congruent, we try to fold the merged model. This process is regarded as another generalization, or learning, step. Instead of complementary– and congruent merging.

Therefore, we also speak of respectively complementary- and congruent learning.

Figure 3.3: Agent topology in the case of complementary merging. DFA models are stitched head-to-tail.
Complementary merging is used when the models are known to represent event-patterns that are successive in terms of workflow in the underlying system. If possible, the individual models of the agents are combined by means of overlapping head-tail nodes. If there are no overlapping head-tail nodes a merge is not possible.

Figure 3.4 provides a sketch of how two DFAs can be combined head-to-tail. If the first DFA has an edge to an end-state on symbol s, and the second DFA has an outgoing edge from its start-state on that same symbol s, the two edges are replaced by a single edge from the source of the first edge to the destination of the second edge. If the destination of the first edge has outgoing edges, these are added to the destination of the second edge.

In general, such a merge step may result in a non-deterministic finite-state automaton, which can be made deterministic by applying a subset algorithm (Hopcroft [74]).

![Figure 3.4: Merging sections of two DFA models by means of stitching.](image)

**Complementary DFA merging**

perform individual learning for DFA₁ and DFA₂

for all possible overlapping end-begin states

- stitch DFA₁, DFA₂
- estimate sample size
- minimize states

endfor

brute force merge-step

**Algorithm 3.3, Collaborative DFA Learning.**
Congruent merging happens when merging DFAs that tend to have similar structures or cover the same event-time-intervals, is done. This happens when agents in parallel process domains combine their models to get a global overview. A congruent merge of two models allows for an enriched model in terms of the level of its details.

Two DFAs from similar environments are merged by combining the start-states of both DFAs. This may result in a non-deterministic finite-state automaton, which again can be made deterministic by applying the subset algorithm. In general, after merging with possible applications of the subset algorithm, the resulting DFA may have a considerable increase of the number of states compared to the original DFAs. It therefore makes sense to attempt to apply a learning step on the resulting DFA.

However, the blue-fringe algorithm is no longer applicable. This is because the combined-DFA contains no information about blue or red states, and the merged DFA is not a DFA in PTA-format. Instead we use a DFA minimization algorithm followed by a brute-force merge-mechanism, knowing, it has a costly $O(n^2)$ time-complexity. DFA minimization (Hopcroft [73]) refers to the task of transforming a DFA into an equivalent DFA which has minimum number of states. The brute force mechanism tries to merge all state-pairs of the DFA, repeatedly, until no more improvements are found, according to the MDL scores.

To carry out this brute-force learning step, we again need an MDL score for the DFA resulting from the merge. Unfortunately, this score is not readily available, because we must assume that the samples from which the original DFAs were learned are not available anymore. We therefore need an estimate of the data-to-model part of the MDL score of the resulting DFA. In the previous section, we discussed how the MDL score is computed using the number of sentences in the
sample (S) and the maximum sample length (L). In particular, the MDL score does not depend on the particular sample, but rather on the sample size (the S).

At first sight, one might think that the original sample sizes could just be added to obtain a new sample size for the resulting DFA, but this is not the case. In congruent DFA merging, there might be overlap in the samples, and in complementary DFA merging, adding the sample sizes makes even less sense. So, we will have to estimate a sample size for the resulting DFA. In fact, assuming that the sample size S and the maximum sample length L are available for the original DFAs, we can from these numbers compute a “sample density” D, which is S divided by the total number of accepted sentences with length less than or equal to L (the N as mentioned in that same section).

\[ D_l = \frac{S_l}{N_l} \]

Now, let S₁ and S₂ be the sample sizes of the original DFAs, and L₁ and L₂ be the maximum sample lengths, and let D₁ and D₂ be the sample densities. An estimate for the maximum sample length L_{merge} of the resulting DFA is obtained as follows:

for congruent learning, we use

\[ L_{merge} = \max(L_1, L_2) \]

for complementary learning we use

\[ L_{merge} = L_1 + L_2 - 1 \]

For the sample density D_{merge} of the resulting DFA we use the average of D₁ and D₂.

\[ D_{merge} = \frac{D_1 + D_2}{2} \]

To compute the estimated sample size S_{merge} of the resulting DFA we compute N_{merge}, i.e. the total number of accepted sentences with length less than or equal to L_{merge} (by recursive backtracking and counting on the merged DFA), and multiply this number by D_{merge}.

\[ S_{merge} = N_{merge} \cdot D_{merge} \]

This computation allows for the computation of an MDL score, and thus allows for the application of a learning iteration on the resulting DFA.
Collaborative DFA learning

perform individual learning for DFA₁ and DFA₂

for all overlapping start-states
merge DFA₁, DFA₂
estimate sample size
minimize states
endfor
brute force merge-step

Algorithm 3.4, Collaborative DFA Learning.

Applying a congruent merge not always means a plain overlay of labels, as can be seen in Figure 3.6. Since the merge is exact, a transition with a label b followed by a transition with a label 2 is in general not allowed. However, depending on the data-to-model part of the MDL score, a possible generalization might allow for it.

Figure 3.6: A congruent merge of the two datasets on the left does in general not result in the one shown in the middle, because the merge is exact. The generalization could allow for it depending on the MDL scores.
3.3 DFA learning in practice

3.3.1 Learnability of languages

Learning languages is one of the fundamental problem areas inside the discipline of machine learning and linguistics. A class of languages is said to be learnable if a learning function exists that can identify any target language in that class from a sequence of sentences from that language. The study on the learnability of languages (Gold [58]) focuses on the possibility of learning a certain class of languages given a specification of the conditions under which learning has to take place.

Learning a grammar by means of finding an automaton that generalizes over unseen data but maintains a low generalization error over the sample data is, in terms of learnability, not a trivial task. Finding grammars of regular languages is classified as a problem which is believed to be of exponential time-complexity\(^{14}\). This means that a grammar induction task takes a very large amount of time for a large amount of sentences. Finding an optimal DFA is proven to be NP-complete (Gold [59]). In the case of learning from positive examples only it is impossible to provably learn the correct (optimal) DFA or, even worse, converging to an approximation of the optimal DFA is found to be an NP hard problem (Pitt and Warmuth [109]).

A second problem we have with DFA learning using MDL is that it has been proved by Adriaans and Vitányi [7] that using MDL to decide on the next merging step of a DFA does not always lead to a better model. From that point of view, MDL will not help us to construct an optimal DFA in terms of a process of incremental compression since calculation of the MDL code cannot give any guidance on its own for compression (or expansion) of a (DFA) model.

Although these theoretical limits exist, quite a few DFA learning algorithms have proven their value. It has been known from experiences that Blue Fringe is the best known DFA Folding algorithm and that MDL as criterion for merging provides good results. In the previous sections we provided a way of using the heuristic methods

\(^{14}\) The computational time-complexity of a task is the number of steps that it takes to complete as a function of the size of the input, using the most efficient known algorithm. E.g. if a task that uses an input of length \(n\) is solved in \(n^2\) steps we say this task has a time-complexity of \(n^2\). Since the exact number of steps depends on what machine or language is being used, it is common to use the big-O notation (the O stands for the "order" of the calculation). One describes the limiting behaviour of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions. Big O notation allows its users to simplify functions in order to concentrate on their growth rates. Formally it is defined as follows: Let \(f(x)\) and \(g(x)\) be two functions defined on some subset of the real numbers. One writes \(f(x) = \Theta(g(x))\) as \(x \to \infty\) if and only if there exists a positive real number \(M\) and a real number \(x_0\) such that \(|f(x)| \leq M|g(x)|\) for all \(x>x_0\).
that try to find good local optima. Lang [82] demonstrated experimentally that the average case is possible within polynomial time. In other words, the Grammar Induction tasks provide in most cases a DFA which is good enough. As a piece of solace, one can consider an analogy of the quick-sort algorithm which is commonly used as the best sorting algorithm: the algorithm has a very good average performance of $O(n \log n)$, though in worst cases performs as bad as $O(n^2)$. In most cases, the conditions on the data are suitable enough to score the average performance. In our DFA learning we simply have to deal with the ‘good enough’ assumption. In chapter 4 we will see that learning patterns from network-events in grid infrastructures results in reasonable models.

### 3.3.2 Quality of merged models

We want to make a critical remark on the value of models as they are obtained by means of a merging. A merged model which is obtained from merging two models of individual learners, each learning a half of a particular dataset, can be worse in the sense of lower quality, than a model that would be obtained by a single learner that learns from the complete dataset: A merged model may have a higher MDL score.

**Theorem 3.1:** A merged model which is obtained from models that have been learned from subsets of data is not always the best model in terms of MDL

**Proof:** (Sketch) We take an example in which two generalized models are merged into one single model. The generalizations hide the details about the original sentences in the datasets which are not available anymore at the moment of merging.

---

15 Sorting algorithms are of fundamental interest in computer science and mathematics. They are of importance to many other algorithms such as search and merge algorithms which are de-facto used in databases. Quick-sort relies on partition array. An element, called a pivot is chosen, all smaller elements are moved in front of the pivot and all greater elements are moved behind it. Efficient implementations of quick-sort are among the fastest sorting algorithms in practice. Together with its modest $O(\log n)$ space usage, this makes quick-sort one of the most popular sorting algorithms. However, the most complex issue in quick-sort is choosing a good pivot element; consistently poor choices of pivots can result in drastically slower $O(n^2)$ performance, but for most cases the median as the pivot seems to be sufficient, resulting in a time-complexity of $O(n \log n)$. 

---

46
Let us take dataset of six sentences, which is split in two halves,
Subset 1: \{abcd, abccd, abcccd\} and subset 2: \{abcd, abcbcd, abcbcbcd\}.

Let us learn DFA models using the algorithms described in the beginning of this chapter. The DFA learned from subset 1 is depicted in (a). The DFA learned from subset 2 is shown in (b). The merged model is shown in (c) whereas the DFA model learned from the combination of both subsets is shown in (d).

The merged model differs from a model that is learned from the whole dataset. The model in (d) has a lower MDL score and is therefore said to be better.

![Diagram](image)

**Figure 3.7**: A merged model is not always the best model

### 3.4 Conclusion

We have described how DFA models can be inferred from (event) data in local and distributed environments. We explained how we merge DFA models. In applying MDL we have discussed how to calculate the model and data-to-model code.

The method described in this chapter is applied in the case environment of computing grids, which will be presented in the next chapter.