Lints, T. (2009). Relation learning with bar charts. In *IEEE Symposium on Intelligent Agents, 2009. IA '09*, pages 77–83. IEEE.

T. Lints, "Relation learning with bar charts," in *IEEE Symposium on Intelligent Agents, 2009. IA '09*, pp. 77–83, IEEE, 2009.

```
@inproceedings{Lints09_BarCharts,
  author = {Taivo Lints},
  title = {Relation Learning with Bar Charts},
  year = {2009},
  booktitle = {IEEE Symposium on Intelligent Agents, 2009. IA
'09},
  publisher = {IEEE},
  pages = {77-83}
}
```

# Relation Learning with Bar Charts

Taivo Lints

*Abstract*— **The paper reports on the work in progress on developing a machine learning method that is inspired on how a human operator might visually try to find relations between a large number of incoming data streams originating from sensors and controllable actuators. A large number of plot-like data structures of various dimensions are periodically generated for the combinations of given data streams and searched for regularities. Controllable variables are manipulated in some random or systematic way until the observed regularities allow for more intelligent behavior (if required). Information about interesting (as defined by the task at hand) relationships is extracted from the data structures, and human-readable control rules for the system are formed. For most nontrivial applications the method can be computationally very expensive, but the potential for parallelizability and for smart optimizations together with the continual increase of capabilities of the off-the-shelf computers suggest that the method will be feasible already in the nearest future.**

## I. Introduction

LEARNING can be seen as a process of discovering and remembering associations. As it is a typical way of achieving intelligent behavior, learning is one of the core topics of research in the field of Artificial Intelligence, having a separate subdiscipline called Machine Learning devoted to finding ways of achieving learning in technological systems, mainly in computers. A large number of various algorithms and techniques for machine learning have been proposed, each of them having their share of advantages and disadvantages. However, achieving learning in machines is far from being a solved problem with only minor technicalities left to figure out. Rather, the field of Machine Learning is wide open to new ideas and, not the least due to the quick increase of performance of computers that keeps making more and more approaches practical, sees a lot of ongoing research activity. In the following paper, a learning method is proposed that is computationally quite demanding, but with some further work might already be practically usable with current technology. The main target area of applications is systems that interact with their surroundings but are not too small to lack computing power. The idea is quite general, though, and can be applied in other areas as well.

Taivo Lints is with the Research Laboratory for Proactive Technologies, Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia (phone: +372 56 625 777; fax: +372 6202 101; e-mail: taivo@taivo.net).

This work was supported in part by Research Laboratory for Proactive Technologies in Tallinn University of Technology, Department of Computer Control in Tallinn University of Technology, Estonian Information Technology Foundation, Estonian Doctoral School in ICT, Estonian Ministry of Education and Research (grant SF0140113As08), and Estonian Science Foundation (grant ETF6182).

Manuscript received Nov 12, 2008.

## II. Description of the Idea

### A. General Description

The research goal of current work is creating a machine learning method that is:

- at all stages easily understandable for (and intervenable by) a human observer,
- general enough to be applicable in different practical areas without major reconfiguration,
- able to extract delayed effects (for example when a lightbulb always lights up 10 seconds after button press),
- able to forget outdated knowledge,
- and suitable for a software-intensive hardware system that has to learn to behave sensibly with minimal initial knowledge about its sensors and actuators.

The approach taken is to look at learning as a process of discovering and remembering associations, considering a case of having a large number of incoming data streams (from sensors) and controllable outputs (to actuators), and thinking about how a human operator might behave so as to learn how these data streams covary with each other and with the controllable outputs. The behavior of the hypothetical human is considered here on the level of explicit acts and conscious reasoning, not on low neuronal level.

A common way of making sense of a big dataset is to represent it in some graphical way. When looking for relations between two variables, the simplest thing to try would be to take a two-dimensional graph, assign one variable to x-axis, the other variable to y-axis, plot all the datapoints, and examine the resulting graph visually for regularities. Given enough datapoints it is relatively easy for a human to detect from the plot the existence of some relationship between two variables, except maybe in some particularly complicated cases. Doing so for every possible pair of given variables would thus enlighten us about most of the one argument functions between them (though not necessarily suggest what exactly those functions are). Finding functions of more than one argument would generally require plotting in more than two dimensions, which can become difficult to grasp visually, unless some clever graphing methods are used.

As of scrutinizing the relationships between (unknown) controls and sensors, the common way for a human operator would be to play around with the controls and observe if that has any effect on sensor values. Apart from the necessity to behave proactively, the rest of the relation finding behavior can be the same plotting procedure described in the previous paragraph.

The machine learning method proposed in current work is directly inspired by this simple and quite intuitive potential human behavior. A large number of plot-like data

978-1-4244-2767-3/09/$25.00 ©2009 IEEE

structures of various dimensions are periodically generated for the combinations of given data streams and observed for regularities. Controllable variables are manipulated in a random, or in some systematically scanning, way (so-called motor babbling) until the observed regularities allow for more intelligent behavior (if required). Information about interesting (as defined by the task at hand) relationships is extracted from the data structures and human-readable control rules for the system are formed.

### B. Related Work

As the proposed method is intentionally derived from the common and intuitive way of human behavior during data analysis, it cannot be novel on the general idea level. However, I am not aware of anybody using online periodical formation and processing of a large number of plot-like data structures as a machine learning technique in the specific way described in current work. It can in part be due to the high computational cost of this method, which in the most simplistic brute force approach would turn out to render the method practically infeasible. However, given the ever faster computers in combination with certain smart ways of reducing the computational complexity, the method looks promising.

There exists a lot of work on analyzing large datasets in the field of Data Mining. However, data mining techniques tend to be not well suited for (semi-)autonomous software intensive hardware systems, or at least require considerable refitting. It is mainly due to the typical use cases of data mining (business intelligence, analysis of data from scientific experiments, etc.) where requirements and allowances differ from embodied software systems – generally a huge data store is available together with excessive computational power and no strict time limits on arriving the results. The most relevant data mining work (with regards to this paper) is probably done on the border between data mining and signal processing where the restrictions on computing time and storage force researchers to invent efficient stream processing algorithms that, for example, dynamically maintain histograms while looking at each incoming datapoint only once and then discarding it (e.g. [1]). A brief review of data stream mining can be found in [2]. The approach in [7], analogously to my work, takes minimal human intervention in sensor stream analysis as a requirement, but their focus is on finding periodic components in individual streams. A system that is able to monitor a large number of data streams in real-time is described in [10]. However, only general statistics (including correlation) is extracted there, not the more detailed relationships as in current work. An approach that uses information from multiple streams in order to estimate missing datapoints is proposed in [9], but their goal is mainly to use that info to enhance estimation of relatively regular and complete time series (where data points are produced at every time step), while in my work there is no restriction for the streams to be interpreted as evolving time series.

System identification and regression analysis achieve quite good results in finding and specifying how system variables relate to each other, but they typically work by trying to fit some predefined model from a limited existing set with observed data. The models available to the identifyer can place considerable limits on what kind of system models can be generated. Artificial neural networks do a good job in nonlinear regression, but their disadvantage is opaqueness – it is difficult for a human observer to quickly understand what a given neural network has learned by looking at the internals of that network.

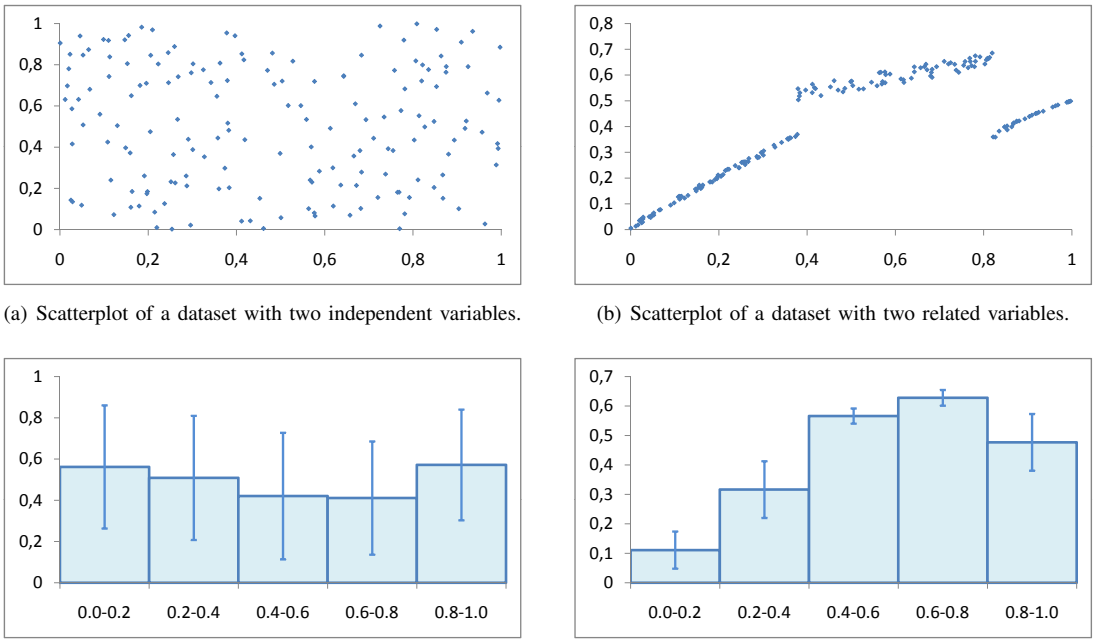
Continuous case-based reasoning [8] and interaction history architecture [5], [4] save a limited number of interesting (as defined by the task of the system) experiences which are typically full recordings over some time period of all data streams, and in case of a reocurring situation retrieve a matching case from the memory and try to repeat it. While both are making use of historical sensor data analysis for system control, they generally do so for the full set of data streams instead of trying to identify the most relevant variables and their relations and using only those for re-achieving the expected result.

In [6] it is reported that information distance (also known as Crutchfield-Rényi Information Metric) is a very good metric for measuring the relatedness of sensory streams to each other, outperforming other measures like 1-norm distance, the correlation coefficient, Kullback-Leibler divergence, Hellinger distance, and the Jensen-Shannon divergence. However, the metric only provides information about the strength of relation, not about what the relation itself looks like, and is thus somewhat difficult to use for deriving control rules from the experience, at least with single variable precision (as opposed to whole experience repetition for which it is used in interaction history architecture [5], [4]).

More abstract (typically logic-based) methods, while often easier to follow by a human observer, tend to have a problem with symbol grounding – connecting the internal symbols with external objects and phenomena they refer to – as it is not trivial to derive usable symbolic representations from incoming data streams and later apply the symbolic results to controlling the actuators in a sensible way.

### C. Detailed Description

*1) The Data Structure:* Probably the simplest way to visualize a set of datapoints of two variables is to draw a scatterplot. As can be seen on Fig. 1 it is quite easy for a human to see if plotted variables are independent or related, and this visual analysis could be automatized as well. However, in computers scatterplots require a lot of memory for storing the datapoints, and a lot of computing power to analyze the plot due to the necessity of taking each point into account (drawing all the points on screen is also time consuming, but given our intent to automate the analysis and to leave the human out of the loop for most of the time, it is not particularly relevant at the moment). And even in case of human visual inspection a scatterplot can become difficult to read in case of large datasets.




(a) Scatterplot of a dataset with two independent variables.

(b) Scatterplot of a dataset with two related variables.

(c) Bar chart with error bars based on the same data as (a)

(d) Bar chart with error bars based on the same data as (b)

Fig. 1. The visual indicators of variable independence are the lack of difference along x axis of the y value distribution, and also the existence of different y values for one x value (though the latter is not always required: if both the mean and dispersion of y are constant everywhere, then y is also highly unlikely to depend on x). Note: these are purely illustrative charts, not generated with the described learning system.

The next simple option is a bar chart where the x-axis is divided into bins (ranges) and the height of each bar reflects the average value of y variable in the corresponding range of x variable. This could also be called a piecewise constant approximation of the relation. However, conversion from scatterplot to bar chart is lossy, as it is not possible to visually differentiate between a bar consisting of many identical y values and a bar consisting of randomly distributed y values with the same mean value. Thus it is useful to also add error bars to the graph, which restores our ability to assess the strength of the relationship (Fig. 1). Loss of information due to the binning of x remains, but is adjustable by bin width, and is generally acceptable for most applications.

Storing and analyzing a bar chart is a lot less expensive: there is only one y value plus error bar size per bin, and a fairly limited number of bins. Also, in case of equal-width bins only the total number of bins and the minimum and maximum value of the x variable is sufficient knowledge on the x axis, as opposed to remembering each datapoint separately. But given our interest in online stream processing there is a need for frequent updating of the charts, which can also include relocation of bin boundaries when a new historically maximal or minimal x value arrives (as we may not initially know the possible ranges of variables). And to fully regenerate a bar chart and its error bars, it is still necessary to have all the datapoints available, and to time consumingly reprocess them.

Luckily, with the cost of some further loss of information, it is possible to instead use a one pass bar charting algorithm that needs to see each data point only once and is still able to update itself as new data arrives. It might well be possible to develop a lot better single pass bar charting algorithms, but for a start the following one that I came up with (and that was partly inspired by the various streaming algorithms for histogram creation in the field of data stream mining) should be sufficient:

- Start with a zero width chart with a fixed number of zero width bins, each containing data about the number of datapoints in that bin ($n$), the arithmetic mean of y value of seen datapoints ($mean$), previous incoming y value ($prevY$), and the average absolute difference of y value from previous y value ($avgAbsDif$) (all initially zero). The latter is a kind of error indicator that is very easily calculable in the one pass mode, as opposed to the common dispersion measures (standard deviation, average absolute deviation, etc.) that are dependent on the mean value of y and can thus be a bit troublesome, though not necessarily impossible, to update without having all the datapoints available [3]. On the other hand, using more common dispersion measures might allow for better analysis in some applications, so it is up to the system designer to choose the best one.
- The first incoming datapoint defines the current location of bins (that still have zero width).
- If a new incoming datapoint $(x, y)$ falls into some existing bin, then increase the datapoint count $n$ in that bin by one, update mean using

$$mean = \frac{mean \cdot (n-1) + y}{n}, \tag{1}$$

update the average absolute difference of y value from previous y value using

$$avgAbsDif = \{avgAbsDif \cdot (n-2) + \\ + abs(y - prevY)\} \cdot \frac{1}{n-1} \quad (2)$$

and finally assign $prevY = y$.

- If a new incoming datapoint falls outside the currently used x range, then create a new chart that has the x range extended up (down) to the new x value, and all bins widened accordingly. Go through all bins of the old chart and transfer the data to the corresponding bins of new chart. If an old bin fully falls into the range of a new bin and the new bin is empty, all data $(n, mean, prevY, avgAbsDif)$ is transfered to the new bin, but if the old bin is partly overlapping with one new bin and partly with another, then assume uniform datapoint distribution along x-axis and split the data from old bin to new bins proportionally with the overlaps (also assume uniform y value distribution along x-axis and thus the data to be put into new bin is $(proportion \cdot n, mean, prevY, avgAbsDif)$, where $proportion$ is the proportion of the old bin that overlaps with the new bin). As new bins are wider than old ones, data from several (parts of) old bins often has to be put to the same new bin. This can be done using

$$meanInNew = (nInNew \cdot meanInNew + \\ + nFromOld \cdot proportion \cdot meanFromOld) \cdot \\ \cdot \frac{1}{nInNew + nFromOld \cdot proportion} \quad (3)$$

The value of previous incoming y, i.e., $prevY$, can be carried over without change (as it is not possible to determine anyway which of the bins has the most recent datapoint). The average absolute difference of y value from previous y value is updated as follows:

$$avgAbsDifInNew = \\ (nInNew \cdot avgAbsDifInNew + \\ + nFromOld \cdot proportion \cdot avgAbsDifFromOld) \cdot \\ \cdot \frac{1}{nInNew + nFromOld \cdot proportion} \quad (4)$$

Then the datapoint count is updated:

$$nInNew = nInNew + \\ + nFromOld \cdot proportion \quad (5)$$

Finally, the new datapoint itself is added to the new chart as described in previous point.

The described algorithm also allows for easy implementation of gradual forgetting of old data points as new ones arrive. Namely, it can be done by just putting a cap on the datapoint counter of each bin, that is, if $n$ reaches some maximal allowed value it will not be increased anymore. This will give each new incoming datapoint a slightly higher weight (compared to the datapoints already accounted for) than it would have normally had, and thus weighs down older values.

The algorithm surely introduces some extra loss of information and inaccuracies in the adaptive widening process of x range, but as the widening procedure typically becomes less and less frequent during the operation of the learning system, the accumulating new data reduces the introduced error. If forgetting mechanism is implemented, the somewhat incorrect values will also be gradually forgotten and finally a fully up to date and accurate-within-possible-limits information is present in the data structure.

The algorithm is also easily generalizable to more than two dimensions – a necessary feature if we need to detect relation functions of more than one variable. In higher dimensions there will still be only one potentially dependent variable, and all the other variables / axes are treated as the x-variable / -axis in the preceding algorithm.

The big concern for computational power expressed in previous paragraphs is due to the fact that we will need a very large number of charts – in case of a brute force approach where all possible charts are generated the total number would be:

$$numOfPossibleBarCharts = n \sum_{k=1}^{n-1} {}^{(n-1)}C_k = \\ = n \sum_{k=1}^{n-1} \frac{(n-1)!}{k!(n-1-k)!} \quad (6)$$

where $n$ is the total number of data streams (variables). The first $n$ in the equation comes from the fact that we need to have each variable to separately appear as the potentially dependent variable (i.e., on the y axis in case of two-dimensional charts). And per each potentially dependent variable we need all the possible combinations of other variables by 1, by 2, etc., on the other axis / axes, thus the sum of combinations in the expression. Note that the total number of possible bar charts is somewhat larger than the total number of possible scatterplots because bar charts do not treat all axes as equal (most are severely discretized, one is not) and therefore having, e.g., variable $v$ on x axis and variable $w$ on y axis yields a principally different 2D bar chart than having variable $v$ on y axis and variable $w$ on x axis (while for scatterplots it would yield the same plot, just diagonally flipped).

Given the approach described later where delayed effects are discovered by using time windows on each real data stream to produce additional streams, the number of in-system data streams is considerably larger than the "real" ones. For example, 10 real sensors, 5 controls and 10 timeslots would result in 150 data streams, and the number of possible charts would be a whopping $10^{47}$ (with the additional problem that each new dimension of a chart exponentially increases required memory and computing power). Luckily, in most applications it is not necessary to consider all the possible combinations, and with some

clever thinking it should be possible to already implement interesting systems with todays off-the-shelf hardware.

*2) Data Preprocessing:* In order to be able to detect delayed effects between variables, it is necessary to chart the relations between current and earlier values of those variables. This can be implemented by keeping recent data points in a buffer and applying time windows to this recent history of each data stream, averaging the contents of each window, and taking the current value of each such window as a new variable. Finding out the appropriate size and number of time windows is not trivial, because increasing their number can produce explosive increase in the number of charts, increasing their size and number requires longer buffers, increasing their size means less accurate data due to averaging, but reducing their number and size would also reduce the ability to detect delayed effects between variables.

As already noted, charting all possible combinations of all data streams would not be practically feasible. Thus it is necessary to consider which combinations to use. Although most possibilities for economizing depend on the specific application (e.g., on the ways we are going to use extracted data, on the configuration of the hardware, etc.), there are also several more general heuristics that provide considerable savings. For example, as it is not possible to affect the past, there is no need to look at combinations where some of the argument ("x-axis") variables are newer than the potentially dependent variable. Also, as the rules for control value generation are usually known (generated by the system itself or given by the human designer), it is not necessary for the learning system to plot the dependencies of control values on other variables (except maybe in very complex systems). Yet another helpful thing to note is that small time shifting generally does not change relationships much, thus it would be redundant to separately consider the relationship between, say, $var_{1,t}$ and $var_{2,t}$ on one chart, and $var_{1,t-1}$ and $var_{2,t-1}$ on another chart.

*3) Extracting Useful Information from Bar Charts:* Assuming that bin widths are appropriate (i.e., do not cause too big information loss due to averaging), it is quite straightforward to assess the strength of relationship in the limits of each bin and on the chart in general, if necessary (by looking at the y value mean and dispersion fluctuations along x-axis, as already described on Fig. 1). After determining the reliability of the bin or chart, getting the specific relationship information is even more straightforward – just reading from the chart that "given such and such values of arguments, the dependent variable has on average this value". That does not give any general mathematical model of the relationship, but is in many applications sufficient anyway.

The exact usage of the accumulated bar charts depends on the type of application. For example if we are dealing with controlling an embodied system, it would be a good idea to extract control rules from the charts periodically instead of doing full time-consuming analysis on each control loop iteration. To generate the rules automatically, it is typically necessary to have some reward / reinforcement signal in the system, and to look at how the value of this signal depends on other variables. The control rules would likely include information about what control values to apply or avoid given the current or recent sensor readings and values of other controls. In addition, the rules can include the reliability information of the relationship that would affect, for example, the probability of applying given rule. Such control rules would also be well understandable for a human, and thus it would be easy to modify, remove, and add the rules by hand if necessary. The specifics of rule generation and usage can be quite intricate and discussing them is out of the scope of current paper.

## III. Experiments

The work on the presented idea is ongoing and started only recently, thus it not possible yet to provide thorough examples of its usage. Therefore, only a brief description of preliminary tests is given here. These examples cannot be taken as a proper experimental proof of the concept. Also, it is most certainly possible to get better results with this learning system on the same test cases after further research has improved the idea. No tests have been conducted yet on real embodied systems – all the following are purely virtual.

### A. XOR

**Test Case:** learning the equivalent of binary exclusive or.

**Setup:** There is one variable controllable by the learning system that can have the value of 0 or 1. The total number of sensors is 3, two of them have random independent values of either 0 or 1, and one is reinforcement signal that is 1 when both other sensors have same value and control is 1, or both other sensors have opposite values and control is 0. Otherwise, reinforcement signal is -1.

The number of time windows is 4, the width being 1 time step. The number of bins on each argument axis of each chart is 4. On charts, only reinforcement signal is considered to be a potential dependent variable, and argument variables are: in two dimensions, all controls one by one (i.e., the current and past 3 values of the control); in three dimensions combinations by two of all controls, and combinations by two of controls and sensors where control value is more recent; in four dimensions combinations by three of controls with one control value per combination and sensors not being more recent than control value. Many charts are clearly redundant for the task, but that is expected in creating a non-application-specific learning system – it must decide itself which are important. Avoidance rules are generated after every 100 time steps.

**Results:** After the first rule generation, no more mistakes are made: control always equals $sensor_0 XOR sensor_1$. The generated avoidance rules have the following form: [['S', 2, 0], -1.0, ['C', 0, 0.0, 0.25], [['S', 0, 0, 0.0, 0.25], ['S', 1, 0, 0.0, 0.25]], which says that if the most recent values of sensors S0 and S1 are between 0 and 0.25, then avoid assigning values between 0 and 0.25 to the control variable. Range is 0 to 0.25 due to the fact that we have 4 bins and the total range of variable values is 1.

### B. Crash Avoidance

**Test Case:** learning to not crash into walls in a rectangular room.

**Setup:** A virtual creature is in a room of 9x9 units and can move around in discrete steps. The number of controls is 2: one causes the creature to step forward one unit if turned on, and the other one to turn either left (value -1), not to turn (0) or turn right (1). Only one of the controls is allowed to be active at each time step (i.e., only step or only turn). The number of sensors is 4: three of them tell whether there is a wall (1) or not (0) correspondingly on the left, in front of, or on the right of the creature; the fourth sensor is reinforcement signal that is -1 if creature tried to step into a wall, and 0 otherwise.

The number of time windows is 10, the width being 1 time step. The number of bins on each argument axis of each chart is 5. The variable combinations being charted are similar to those described in XOR test, with the addition to four dimensions the combinations by three of two controls and one sensor (the latter not being more recent than the oldest control variable), and the combinations by three of two controls and one sensor where the sensor variable is timewise between the two controls.

The creature wanders around randomly. Avoidance rules are generated after every 100 time steps.

**Results:** After the first rule generation, no more crashes occur. An example of resulting avoidance rules is: [['S', 3, 0], -1.0, ['C', 0, 0.8000000000000004, 1.0], [['S', 0, 0, 0.8000000000000004, 1.0]], which says that if the most recent value of sensor S0 is between 0.8 and 1, then avoid control C0 between 0.8 and 1, and it means that if there is a wall ahead, do not step forward (this meaning is obviously only assigned by the human observer, not something the virtual system would understand).

### C. Keeping Close to a Light Source

**Test Case:** learning to go towards a virtual light source.

**Setup:** A virtual creature is in a room of 19x19 units and can move around in discrete steps. The number of controls is 1: the creature will make one step to North if the control is 0, to East if 1, to South if 2, to West if 3. The number of sensors is 6: four of them indicate the presence of a wall in the corresponding direction (North, East, South, West), one gives the angle in radians towards light source relative to the x-axis originating from the creature ($atan2(posOfLightY - posOfCreatY, posOfLightX - posOfCreatX)$), and one is a reinforcement signal that is 1 if the distance from creature to light decreased, -1 if increased, and 0 if remained the same.

The number of time windows is 3, the width being 1 time step. The number of bins on each argument axis of each chart is 15. On charts, only reinforcement signal is considered to be a potential dependent variable, and argument variables are: in two dimensions, all controls one by one; in three dimensions combinations by two of all controls, and combinations by two of controls and sensors where control value is more recent. Avoidance rules are generated after every 100 time steps, and are executed probabilistically to allow for some exploration even after rules have been generated: with probability 0.7 the avoidance rules are followed, otherwise the random control value is applied without appropriateness check.

**Results:** if the light source is put into a corner of the room, then after the first rule generation the creature will efficiently move to that corner. However, if the light source is in the middle of the room, the current system has difficulties learning to keep to its vicinity. The reason is likely to lie either in unsuitable parameters of the learning system or in the primitivity of rule following system, and fixing it will require some further analysis (which is not undertaken yet due to the project being in the initial stage).

## IV. Future work

This paper is mainly only a conceptual overview of the idea. Considerable further work is needed before it can be successfully applied on interesting and / or serious problems. The topics needing further research include:

- How to generate bar charts computationally more efficiently.
- How to select the combinations of variables to look at, and how to automate that selection (e.g., if no relationship is detected between some variables, then given chart might be removed from the learning system, either permanently or for some period).
- How to select the number and size of time windows in data preprocessing.
- How to analyze the charts computationally more efficiently.
- How to extract only the most important information from the large number of charts, so as to avoid information overload in the control system.
- How to generate the rule set so as to cover enough various situations the controllable system might find itself from, not only the rules for worst or best cases as the naive "let's look only at situations with highest and lowest reinforcement values" approach might do.
- Which procedures to use for applying the found rules so as to make use of the gathered information most efficiently.
- How to balance exploration and exploitation, i.e., how much and which kind (random, systematic) exploratory action is needed for the bar charts to accumulate diverse enough data for the system to behave well, as opposed to settling down quickly to potentially far-from-best control rules.
- How to adaptively change various parameters during system's work (e.g., varying the width of individual chart bins so as to represent more important regions on the axis more accurately).
- How to parallelize the system in order to have more computing power available for stream analysis and rule generation (it seems that the system is inherently quite decomposable – operations on one chart generally do not interfere with operations on other charts; also,

given some initial knowledge about the sensors it might be possible to group similar sensors to modules, add complexity-reducing pre- and intermediate processing, introduce hierarchical organization of data flows, etc.). This is likely to be very important for all practical applications except the simplest toy problems, as the amount of required computing power tends to increase explosively with the addition of sensors, actuators, and time windows on memory buffers.

- A proper analysis of what the method can and cannot do.
- A proper comparison with other machine learning methods, including on the basis of theoretical capabilities, computational cost, and suitability for various applications.

## V. Conclusion

The machine learning method inspired by human behavior during data analysis can potentially produce good results, especially in software-intensive embodied systems, and at the same time the internals of this learning system are easily understandable to a human observer at all stages, which makes it easier to actively intervene with system operation as necessary, to transfer the learned knowledge to other systems, and to develop new more complex approaches based on this method. Although the method in its most primitive form does not scale to cope with problems of any practical interest due to the extremely sharp growth of computational requirements, it nevertheless looks promising because of its potential parallelizability, the possibilities for smart optimizations, and the fast growth of capabilities of computer technology.

## VI. Acknowledgement

Thanks to Leo Mõtus for providing me the possibility to do highly unconstrained research. Thanks to the three anonymous reviewers for reading the first version of this paper and providing constructive criticism.

## References

[1] C. Buragohain, N. Shrivastava, S. Suri, Space efficient streaming algorithms for the maximum error histogram, in: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, 2007.

[2] M. M. Gaber, A. Zaslavsky, S. Krishnaswamy, Mining data streams: A review, SIGMOD Rec. 34 (2) (2005) 18–26.

[3] M. Hoemmen, Computing the standard deviation efficiently, `http://www.cs.berkeley.edu/~mhoemmen/cs194/Tutorials/variance.pdf` (2007).

[4] N. A. Mirza, Grounded sensorimotor interaction histories for ontogenetic development in robots, Ph.D. thesis, University of Hertfordshire (2008).

[5] N. A. Mirza, C. L. Nehaniv, K. Dautenhahn, R. te Boekhorst, Grounded sensorimotor interaction histories in an information theoretic metric space for robot ontogeny, Adaptive Behavior 15 (2) (2007) 167–187.

[6] L. Olsson, C. L. Nehaniv, D. Polani, Measuring informational distances between sensors and sensor integration, in: Artificial Life X, 2006.

[7] S. Papadimitriou, A. Brockwell, C. Faloutsos, Adaptive, hands-off stream mining, in: VLDB '2003: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB Endowment, 2003.

[8] A. Ram, J. C. Santamaria, Continuous case-based reasoning, Artificial Intelligence 1-2 (1997) 25–77.

[9] B.-K. Yi, N. D. Sidiropoulos, T. Johnson, A. Biliris, H. V. Jagadish, C. Faloutsos, Online data mining for co-evolving time sequences, Data Engineering, International Conference on (2000) 13.

[10] Y. Zhu, D. Shasha, Statstream: Statistical monitoring of thousands of data streams in real time, in: VLDB '02: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB Endowment, 2002.